USING WEB SERVICES TO INTEGRATE DATA AND COMPOSE ANALYTIC TOOLS IN THE LIFE SCIENCES

by

ZHIMING WANG

(Under the Direction of John A. Miller)

ABSTRACT

Advances in technology and computational approaches have resulted in an explosive increase in the quantity of biological data. How biologists share data and analytical tools efficiently is becoming a fundamental issue. One of the promising technologies to handle this challenge is Web service technology, which provides advanced features such as language independence, platform independence, compliance with universal standards and decoupling of service from client.

At the end of 2007, there were 1078 biological databases. Providing biologists central and uniform access to all types of data stored in biological databases is becoming critical. To minimize disruption of current operations, maintain local autonomy and handle heterogeneities, federated databases and Web services have been proposed as a viable solution. This dissertation explores this situation and reports on our experience with testing multiple approaches for biological database integration. It discusses the trade-offs among performance, support for heterogeneity, robustness and scalability. Of significance is the discovery that the most flexible approach, Web Services, performs very competitively.

Given the increasing prevalence of Web services that access biological data from multiple different locations and databases, we have seen an increasing interest in biological Web service

composition to perform complex bioinformatics tasks. Although some research on composing biological Web services has been performed, resulting in tools such as BioMoby and Taverna, these tools are still too difficult to be easily used by the average biologist. Therefore, lowering the learning curve for Web service composition is a critical need. With this objective in mind, we have designed and implemented WS-BioZard, a new and comprehensive framework using multiple technologies and semi-automatic service composition to address this need.

INDEX WORDS: Database Integration, Federated Databases, Data Warehousing, Web Services, Web Service Composition, Ontology, Semantics, SAWSDL, BPEL.

USING WEB SERVICES TO INTEGRATE DATA AND COMPOSE ANALYTIC TOOLS IN THE LIFE SCIENCES

by

ZHIMING WANG

B.S., Naval Institute of Engineering, P.R. China, 1995

A Dissertation Submitted to the Graduate Faculty of the University of Georgia in Partial Fulfillment of the Requirements for the Degree

DOCTOR OF PHILOSOPHY

ATHENS, GEORGIA

2008

USING WEB SERVICES TO INTEGRATE DATA AND COMPOSE ANALYTIC TOOLS IN THE LIFE SCIENCES

by

ZHIMING WANG

Major Professor: John A. Miller

Committee: Jessica C. Kissinger
Eileen T. Kraemer
Khaled Rasheed
Alan R. Gingle

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
JUNE 2008

DEDICATION

I dedicate this dissertation to my beautiful wife for her unconditional support throughout and beyond this graduation work, and to my wonderful sons Ben and Dan - my source of joy and energy that kept me moving forward. A special feeling of gratitude goes to my loving parents and parents-in-law whose love and support have never left my side. I have always received special encouragement from my uncle Huamin. Thank you for being there for me throughout my entire doctorate program.

ACKNOWLEDGEMENTS

I wish to thank my major advisor and mentor, Dr. John A. Miller for his countless hours of advising, reading, encouraging, and most of all patience throughout the entire process. The program was one of the most important and formative experiences in my life. Without your guidance and help every step of the way, I could not be where I am today.

I am also very grateful to Dr. Jessica C. Kissinger, Dr. Eileen T. Kraemer, Dr. Khaled Rasheed, and Dr. Alan R. Gingle. Thank you all for agreeing to serve on my committee and offering me time and stimulating discussion about my research. Especially, I need to express my gratitude and deep appreciation to Dr. Jessica C. Kissinger for the continued support both financially and professionally. You helped me keep perspective on how to better my overall research and presentation skills.

I must acknowledge as well Mark S. Heiges for providing tremendous amount of assistance with the technical development, and Cristina Aurrecoechea for generously giving her time and suggestions during my paper writing process. I also need to thank the many friends I meet in the ApiDB project and LSDIS lab, beginning with Xin Gao, Congzhou He, Abhijeet Bakre, Cary Pennington, Kathy Jones, Kelly Storm, Adhemar Zerlotini, Doug Brewer, Rui Wang, Sadiq Charaniya and Shefali Shastri. Their excitement and willingness to provide feedback made this research an enjoyable experience.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER 1

GENERAL INTRODUCTION

Over the last few decades, the volume of bioinformatics data has increased dramatically. The tremendous volume and diversity of biological data greatly improve the biologists' ability to study the interactions between the components of a biological system and how these interactions give rise to the function and behavior of that system. Such studies need access to multiple types of data which are likely to be stored in different, geographically distributed databases. For example, in order to answer the following questions, biologists need access to different data stored in different databases:

"Find all single nucleotide polymorphisms for TP53 gene in human and mouse orthologs with epidemiologic assessments of cisplatin treatments in adenocarcinoma cells."

"Find all structural homologs of putative N-acetyltransferase SR144 in B. subtillus in SCOP family Acyl-CoA N-acetyltransferase." [1]

Providing biologists with central, uniform access to all types of data is not a trivial task. It would be a poor solution, if not impossible, to make a single database to include all biological data [2]. To provide better query facilities and expedite the research process in an automatic way, database integration is essential, and is one of the most important bioinformatics research areas [3]. In chapter two, multiple database integration approaches are investigated for the Apicomplexean Database (ApiDB) project, based on the criteria of system performance, data replication, autonomy, system scalability/maintainability and schema evolution.

Because Web service technology provides advanced features such as platform independence, language independence, and adherence to universal standards, it can play an important role in database integration. In a complementary fashion, Web service technology also can seamlessly incorporate software to analyze bioinformatics data. While individual bioinformatics Web services are useful, situations often arise where more than one service is required to perform a complete biological analysis. For example, whenever a new nucleotide sequence is annotated, biologists try to understand its functionality via searches for homologous sequences. This task can be achieved through composing biological Web services. First a BLASTX[1] Web service could be used to identify amino acid sequences from a well-known protein sequence database. Then another Web service is used to retrieve the 'good hits' amino acid sequences and yet a third protein domain Web service is used to identify protein domains.

Two main approaches exist for composing bioinformatics Web services: 1) specific tools for composing bioinformatics Web services and 2) general tools based on the Web Service Business Process Execution Language (WS-BPEL). Compared with specific tools such as Taverna and BioMoby, BPEL tools are widely supported by industry because many major industrial companies such as IBM, Microsoft, Oracle, and BEA worked together to produce the BPEL specification. Hence, more editors and tools are available such as ActiveBPEL, OracleBPEL, NetBeans BPEL and Eclipse BPEL. Furthermore, BPEL tools are reusable since the BPEL process itself is a standard Web service. However, BPEL editors are hard for biologists to use, because of the complexity of control flow and data flow in BPEL. The complexity of control flow is mainly due to the specification of WS-BPEL and the BPEL editor reflects this complexity

1 The Basic Local Alignment Search Tool (BLAST) can be used to infer gene family information and sequences' relationship, such as function or evolution, through comparing the querying sequence to sequence databases and calculating the statistical significance of matches. BLASTX is a special BLAST program to search protein databases using a translated nucleotide query.

in the interface. The complexity of data flow comes from the fact that data handling is done using XPath as well as the inherent data heterogeneity existing in bioinformatics data. Other impediments for biologists trying to use current BPEL tools include generating the process WSDL file from the BPEL process and deploying the BPEL process to a BPEL execution engine.

In response to the complexity of current BPEL tools, we developed WS-BioZard, which provides a comprehensive framework for semi-automatic Web service composition. It includes simplified handling of control flow as well as effective data mediation to reduce the complexity of data flow. In this framework, multiple technologies are used to lower the complexity of Web service composition, Web service discovery, and Web service deployment. WS-BioZard utilizes Semantic Annotations for WSDL (SAWSDL:*www.w3.org/2002/ws/sawsdl/*) and domain ontologies to add semantics to Web services. This allows us to design and implement (1) a service discovery tool with a sophisticated User Interface (UI), (2) a multi-faceted UI Wizard that facilitates the data mediation and service composition tasks for users, and (3) a simplified BPEL editor focusing more on the business logic, while keeping the most useful capabilities of WS-BPEL 2.0. The result of the composition is a BPEL executable that can be automatically deployed into a BPEL execution engine and invoked as a standard Web service.

CHAPTER 2

AN EVALUATION OF MULTIPLE APPROACHES FOR FEDERATING BIOLOGICAL DATA[2]

2. Zhiming Wang, Xin Gao, Congzhou He, John A. Miller, Jessica C. Kissinger, Mark Heiges, Cristina Aurrecoechea, Eileen T. Kraemer and Cary Pennington. Submitted to *the Journal of Information Technology Research*. Corresponding author, 1/23/2008. Address: Department of Computer Science, University of Georgia, Athens, GA 30602. E-Mail:jam@cs.uga.edu. Telephone: 706-542-3440. Fax: 706-542-2966.

1. ABSTRACT

Between technological breakthroughs and new computational approaches, the quantity of biological data is increasing explosively. As of 2007, there were 1078 biological databases. To provide biologists central and uniform access to all types of data stored in biological databases is becoming critical. To minimize disruption of current operations, maintain local autonomy and handle heterogeneities, federated databases and Web services have been proposed as a viable solution. This paper explores this situation and reports on our experience with testing multiple approaches for biological database integration. It discusses the trade-offs among performance, support for heterogeneity, robustness and scalability. Of significance, is the discovery that the most flexible approach -- Web Services, performs very competitively.

2. INTRODUCTION

The quantity of biological data is increasing explosively; most of these data are stored in databases including the International Nucleotide Sequence Database Collaboration (DDBJ, EMBL and GenBank) and numerous other specialized databases, such as PROSITE, EC-ENZYME, GDB, Reactome, UniProt, PIR, DIP, Pfam, PDB. At the end of 2007, there were 1078 biological databases (Galperin, 2008). This tremendous diversity of biological data greatly improves the biologists' ability to study the interactions between the components of a biological system, and how these interactions give rise to the function and behavior of that system. Such studies need to access multiple types of data which are likely to be stored in different, geographically distributed, databases. However, providing biologists with central, uniform access to all types of data is not a trivial task. It would be a poor solution, if not impossible, to make a single database to include all biological data (Stein, 2003). To provide better query facilities and

expedite the research process in an automatic way, data integration is essential, and is one of the most important bioinformatics research areas (Stevens et al., 2001). There are three main layers of data integration. The bottom layer is the database and database management system (DBMS), in the form of distributed databases, multi-databases, data warehouses or federated databases. The middle layer is software that supports the distributed applications and includes Service-Oriented Architecture (SOA), Common Object Request Broker Architecture (CORBA), Distributed Component Object Model (DCOM) and Java Remote Method Invocation (RMI). The upper layer puts responsibility for integration into the application and includes Link Integration or Query-Based Integration (Stein, 2003; Karp, 1995). Based on Peter Karp's four approaches (Hypertext Navigation, Data Warehouse, Unmediated MultiDB Queries and Federated Databases) (Karp, 1995) and Lincoln Stein's three approaches (Link integration/Web Services, View integration, Data Warehousing) (Stein 2003), one could argue that there are five basic data integration approaches in use: Link Integration, Query-Based Integration, Data Warehouse Integration, Federated Database Integration and Web Service Integration (Stein, 2003; Karp, 1995) as briefly described below. Among them, Federated Database and Web Services are prominent technologies because each minimizes the disruption of current operations, maintains local autonomy, and handles heterogeneities as well as scalability.

- **Link Integration** means that end users get comprehensive and relevant information through hyperlinks from the first data source that end users begin to search. Although it is very successful, it faces problems such as the vulnerability of naming clashes, ambiguities, and the instability of volatile hyperlinks maintained by different hosts (Stein, 2003). The Sequence Retrieval System (SRS) (Zdobnov et al., 2002) is a variation of the Link integration approach. It is designed to retrieve information stored in multiple hosts using the

language ODD (Etzold and Argos, 1993). It stores the data indexes in a global repository, while leaving the actual data in their own repository. The disadvantage is that the integration and data location are not transparent to end users (Stein, 2003; Karasavvas et al., 2004; Etzold and Argos, 1993).

- **Query-Based Integration** means that end users can retrieve information from multiple repositories that can be databases or flat files through a single query. In this approach, the query is often not SQL. The main problems with this approach are the complexity of queries, the fact that a new query language or language extension must be learned and the fact that integration and data location are not transparent to end users. For example, in BioKleisli (Davidson et al., 1997) using a query language called "Collection Programming Language (CPL)" (Peter et al., 1994), end users need to manually choose the database and specify how to use it. Other examples of query-based integration include Object-Protocol Model (OPM) (Chen and Markowitz, 1995; Markowitz et al., 1999) that uses OPM*QL as the query language and P/FDM (Kemp, 2002) that uses Daplex as the query language.
- **Data Warehouse Integration** replicates or summarizes data from remote/component databases to a large central warehouse. Problems with this approach include data synchronization as well as scalability. The next section will cover the pros and cons of Data Warehouses in greater detail.
- **Federated Database Integration** is more flexible when compared with Data Warehouses because no data replication is required (unless for performance or reliability). This obviates the requirement to keep the data in the central repository up to date. All the queries are initially sent to and dealt with at the central site to provide end users with uniform and central access to retrieve data.

- **Web Service Integration** has the potential to become a widely applicable integration approach. Web Service technology is becoming more popular as it matures. This will be discussed in detail in section 4.

The focus of this paper is on the architectural design, implementation and testing of a Federated Biological Database via multiple approaches. We have implemented and evaluated four approaches for the ApiDB project (a member of the National Institute of Allergy and Infectious Disease's (NIAID) Bioinformatics Resource Centers (BRC) *http://www.niaid.nih.gov/dmid/genomes/brc/*). The goal of the ApiDB project (see: *http://www.apidb.org*) is to provide biologists with uniform, integrated and centralized Web access to apicomplexan genome resources that currently are provided by three distinct databases, CryptoDB, PlasmoDB and ToxoDB [3], which represent the apicomplexan parasites *Cryptosporidium, Plasmodium* and *Toxoplasma,* respectively. ApiDB and CryptoDB are hosted by the University of Georgia, while PlasmoDB and ToxoDB are hosted by the University of Pennsylvania. The ApiDB project uses two main software modules separated by functionality, a front-end module that provides uniform Web access for end users via the Web Development Kit (WDK) (http://www.gusdb.org/wdk/) and a data storage module based on the Genomics Unified Schema (GUS) (Davidson, 2001).

As a follow-up to the paper of "A Comparison of Federated Databases with Web Services for the Integration of Bioinformatics Data", we present additional approaches for the integration of bioinformatics data with substantially more detailed information. We consider common database integration approaches to bottom layer of integration in section 3, such as data

3 Historically three component databases/organisms and now five, including two new organisms: *giardia lamblia and trichomonas vaginalis*

warehouses and federated databases and demonstrate why we chose federated databases for our project. In section 4, Web services and Java RMI as middle layer approaches for data integration are discussed. In section 5, we introduce a comprehensive case study, creating a federated database as part of the ApiDB project, including its goals, architecture and requirements. In section 6, multiple implementation of four of these approaches (Database Link, Java Database Connectivity, Web services and Java Remote Method Invocation) are described in detail. Timing and memory usage results for all these implementations are compared and discussed in section 7. Finally, section 8 presents our conclusions, discussions and possible future work.

3. DATABASE INTEGRATION APPROACHES: BOTTOM LAYER OF DATA INTEGRATION

Conceptually, it might be nice to provide biologists with central, uniform access to all the available data via a single database containing all biological data. The difficulty with such a solution is the many profound variations existing in biology such as the naming conventions, the format of sequences, exceptions on most rules, multiple sources and different explanations of the same data, natural variation and experimental error, *etc*. (Karasavvas et al*.,* 2004). Consequently, biologists blessed with such a huge amount of available information from different bioinformatics and functional genomics projects are often confronted with the problems of having to manually identify proper databases and integrate query results when their tasks involve multiple sources (Philippi, 2004). To provide better query facilities and expedite the research process, database integration as the bottom layer of data integration is an essential technology and has been one of the most important bioinformatics research areas (Stevens et al*.*, 2001).

The goal of database integration is to provide end users with uniform, integrated and central access to all types of biological data stored in heterogeneous databases by means of decomposing distributed queries[4] into component queries, issuing those queries, and delivering the combined results into a uniform format for end users (Karasavvas *et al.,* 2004; Wilhelm, 2000; Sheth and Larson, 1990). There are many database integration approaches; among them are Data Warehouses and Federated Databases. Currently, these are the two most common approaches for database integration.

Data Warehouses deal with data integration through conversion according to predefined rules, and then centralize all the data to one data warehouse. This system can retrieve data from any resource to the central repository as long as there is a suitable data conversion/wrapper available. Data Warehouses provide several benefits such as increasing data consistency, improving performance, supporting centralized and integrated access to multiple databases. One issue however, is how to keep the data in the central repository up to date, especially when the data in component sites is updated frequently. In addition, great effort is required to deal with data conversion and database management (Stein, 2003; Philippi, 2004; Schonbach et al., 2000). Since Data Warehouses require data from component databases to be stored centrally, it can cause problems such as scalability and data privacy. The Integrated Genome Database (IGD) project (Ritter et al., 1994) was a Data Warehouse approach that tried to integrate more than a dozen genomic databases (including GenBand and the Genome Database (GDB)) and experimental resources, using their own Concise Object Query Language (COQL) as a query language. Since each source database changed the data model twice a year on average, this requires the IGD to stop service and rewrite the data conversion program every two weeks.

4. Not all database integration needs distributed query, for example, Data Warehouse will not need distributed query because all the data has already been centralized.

Such a schedule is nearly unmanageable and finally lead to the failure of IGD (Stein, 2003). Other Data Warehouse projects include LIMBO (Philippi, 2004) and GIMS (Michael et al., 2003). Considering the vast amount and diversity of data in each component database within ApiDB and the required periodic updating of component databases, it would be difficult to keep the central store up to date. Consequently, Data Warehouses alone as a database integration strategy may not be the best choice for ApiDB, particularly as it continues to grow.

Compared with Data Warehouses, Federated Databases (Sheth, 1990) reduce problems such as keeping a central data store up to date, scalability and data privacy, because there is no need to centralize data from component databases. A Federated Database consists of multiple component databases that are capable of sharing data through a unified interface; thus to end users, it appears as one logical database. In bioinformatics, database federation often means that a user poses one single query and is provided with an answer based on information from across multiple biological databases bound in the federation. The process of accessing discrete databases for an answer to the query is kept transparent to the user (Haas et al., 2002). According to Casey, Federated Databases are becoming a mainstream approach for biological database integration and there are more and more federated databases in the life sciences community (Casey, 2006). The following are examples of Federated Databases:

- iProClass, which integrates protein information such as family relationships, structural and functional classifications (Hongzhan et al., 2003) (see: *http://pir.georgetown.edu/iproclass/*).
- Comparative Mouse Genomics Centers Consortium (CMGCC), which provides “resources in transgenic and knockout mouse models based on single nucleotide polymorphisms (SNPs) in human environmentally responsive genes” (Tyson, 2005) with the data disparately

located in several university research centers (see: *http://www.niehs.nih.gov/cmgcc/dbmouse.htm*).

- Cell Centered Database (CCDB), which provides integrated, rich 3-D structural data ranging from molecules to cells, with a central database based on Oracle 8i and using Storage Resource Broker to query the distributed data (Martone et al., 2004) (see: *http://ccdb.ucsd.edu/about.html*).
- Kleisli/K2, developed at the University of Pennsylvania that is a distributed query language aimed to be used in Federated Databases. It depends on data drivers to communicate with distributed component data sources (Davidson et al., 2001).

Component databases in a federation can be characterized by heterogeneity, autonomy, and distribution (HAD) (Sheth and Larson, 1990; Haas, 2001). Heterogeneity, autonomy, distribution and scalability/maintainability are the main concerns in database federation.

- **Heterogeneity** includes structural heterogeneity and semantic heterogeneity. Both heterogeneities are important issues in biological data integration. Many different data models including relational and object-oriented are used in biological databases. Some data are even stored in flat files. Semantic heterogeneity can be more severe and is caused by a lack of a consistency in naming conventions in the life sciences.
- **Autonomy** includes design autonomy, communication autonomy, execution autonomy and association autonomy (Sheth, 1990). It would be neither feasible, nor desirable, to establish total centralized control of all the biological data sources existing around the world.
- **Distribution** is defined as data being distributed among multiple component databases, which could be geographically located in different places, but connected by a network.

Data distribution can bring in the benefits such increased availability and reliability as well as improved access times.

- **Scalability/maintainability** is defined as the system being easily enlarged to fit new requirements, which also aid the maintainability. Considering the rapid increase in biological data and databases, scalability and maintainability are also important issues for biological data integration.

4. WEB SERVICES AND JAVA RMI APPROACHES: MIDDLE LAYER OF DATA INTEGRATION

4.1. WEB SERVICES APPROACHES

The Service-Oriented Architecture (SOA) is a software architecture which decouples a service provider from a service consumer. That is, consumers can choose any service from any provider as long as the interface is compatible, no matter what language the service is written in, or what platform the software service is running on. This decoupling brings two important benefits: abstraction and extensibility. Because of abstraction, both the client and server can evolve separately as long as they conform to the interface of the services. A service provider can change the service implementation without affecting the client as long as the interface is kept stable. In addition, a service requester can choose the best service provider or compose multiple services.

Web service technology is the most popular implementation of the SOA. According to the W3C Web Services Architecture Working Group, “A Web service is a software application

identified by a URI, whose interfaces and bindings are capable of being defined, described, and discovered as XML artifacts. A Web service supports direct interactions with other software agents using XML based messages exchanged via internet-based protocols" (Champion, 2002). The basic Web service architecture includes three parts connected over a network: Service Provider, Service Consumer and Discovery Agency. A Service Provider defines its service in the Web Services Description Language (WSDL). A Discovery Agency is used to store the information about Service Providers, for example, the category and location of Service Provider, the interfaces of service invocations, *etc*. After a Service Provider registers its service to a Discovery Agency using a Universal Description, Discovery, and Integration (UDDI) registry, a Service Consumer can query for services from a Discovery Agency also using UDDI, and then connect to the Service Provider to consume the service using the Simple Object Access Protocol (SOAP). Beyond these three parts, there are advanced functionalities such as monitoring, security and composition of Web services. Web services provide benefits such as platform-independence, language-independence and provider-independence (Champion, 2002; Gottschalk et al., 2002; Sheth and Miller, 2003). Compared with federated databases, Web services provide more flexibility in data integration by combining XML with this very loose coupling between client and server. Web services can be used to deal with integrating not only structured data stored in database, but also semi-structured or unstructured data.

Two main obstacles for data integration are structural heterogeneity and semantic heterogeneity. Structural heterogeneity includes different schema or data models, such as structured (relational database, objected-oriented database), semi-structured (web pages, XML) or unstructured (text file, hypermedia, binary objects). Semantic heterogeneity is caused by disagreement about the meaning, interpretation, or intended use of the same or related data

(Sheth and Larson, 1990; Nagarajan 2006). XML as well as additional Semantic Web languages can help resolve both structural heterogeneity and semantic heterogeneity.

- Data Model: XML is an ideal candidate to provide a unifying data model in data integration systems (Arumugam and Chakrapani, 2002) for the following reasons: XML is widely accepted as a data exchange format and recommended as a standard by W3C. XML is platform, language and architecture independent with standard query language such as XQuery and XPath. XML can effectively facilitate data integration through Web services because it is also the format of SOAP.
- Schema Integration and Semantic Heterogeneity: Schema integration is a necessity in many application domains, such as data integration, database federation, and data warehousing. The goal of schema integration is to create a global view, by mapping away the structural and terminological differences between the component sites. Great effort has been devoted to this research area (Batini et al. 1986; DOAN et al. 2001; Elmagarmid and Pu, 1990; LARSON et al. 1989; Parent and Spaccapietra, 1998). The main schema integration approaches include rule-based and machine-learning based, both of them can benefit from XML as the unifying data model. At the same time, semantic heterogeneity needs much consideration during schema integration. Without semantics, there is no way to identify if two terms with the same spelling mean the same thing or not, which can be addressed by the use of ontology. The XML Schema Definition (XSD) contains some useful semantic information which can improve data federation. Ontologies can be used in Web service-based federation to offer a solution to schema integration and semantic heterogeneity problems. Along these lines, the Semantic Annotations for WSDL (SAWSDL, see: *http://www.w3.org/TR/sawsdl/*) which evolved from WSDL-S (see:

http://www.w3.org/Submission/WSDL-S/) can further facilitate Web service federation through adding more semantics to WSDL and XSD.

Web Services performance is a concern for any project. One result discovered by our implementations is that the performance of Web services is very competitive, which will be shown in the performance comparison section. The performance overheads involve the server's hardware capacity, SOAP message size and complexity, XML parser, costs of serialization and deserialization, costs of connection establishment, security validation, UDDI registration and querying, and binary XML vs. text. In certain cases, parsing SOAP messages and data binding cost the most among overall time, especially parsing SOAP messages which occupies 20% to 35% of the whole processing time (Gunther, 2003). Even with more efficient XML parsers, it still makes up a significant portion of Web Service processing. Besides, the larger the message, the longer it takes to parse. The more complex the message, the longer it takes to do data binding. There are several approaches to improve Web service performance that are explained on the following Web page (*http://www.cs.uga.edu/~zhiming*/), which are also the concerns of our project.

Because of the flexibility and extensibility provided by Web services, more biological databases/projects are providing Web services, such as the National Center for Biotechnology Information (NCBI) (Wheeler et al., 2000), the European Molecular Biology Laboratory (EMBL) (Wang et al., 2002) and the DNA Data Bank of Japan (DDBJ) (Miyazaki et al., 2003). The BioMOBY project (MD Wilkinson and M Links, 2002) and MyGrid (Robert et al., 2003) not only provide Web services, but also provide a Discovery Agency function, which lets other Service Providers register their Web Services and lets Service Consumers query for suitable

Service Providers. At the same time, more biological ontologies are emerging. For example, the Gene Ontology (GO: *http://www.geneontology.org/*) provides a controlled vocabulary to describe gene and gene product attributes in any organism. The Plant Ontology (PO) aims to develop, curate and share structured controlled vocabularies (ontologies) that describe plant structures and growth/developmental stages. The Open Biological Ontologies (OBO: *http://obo.sourceforge.net/#rel*) foundary is an umbrella Web address for many well-structured controlled vocabularies and ontologies for shared use across different biological domains.

4.2. JAVA RMI APPROACH

Java RMI is the Java distributed computation model that allows a remote object's method to be invoked by another Java Virtual Machine (JVM). It is a Java specific distributed programming technique. The main purpose of implementing Java RMI is to provide a performance baseline for Web services, both are distributed programming approaches with different data exchange formats and different engines, which may partially explain the performance difference between them. A more detailed performance discussion will be given in section 7.

There are three main components in Java RMI: client, server and RMI registry. Usually the client and server are located on different JVMs, and the client invokes methods on the remote server. First the server should register with Java Naming and Directory Interface (JNDI) so that it can be discovered by the client. Then, once running, the server stays in memory, waiting for a call from the client. Web services are the most generic distributed computation technology, which is to say, one can use any language to implement the server side and client side. RMI requires that both the server side and client side be implemented in Java. While more restrictive than Web services, this requirement has the advantage of dynamic code loading on the server,

meaning the server can download class definition if it is not there already. Since the actual class is transferred by Java serialization, the server can extend new functions dynamically and change behavior accordingly.

5. CASE STUDY: APIDB PROJECT GOALS AND REQUIREMENTS

The goal of the ApiDB project is to provide biologists with uniform, integrated and centralized Web access to apicomplexan genome resources that currently are provided by CryptoDB, PlasmoDB and ToxoDB. All these databases use the Genomics Unified Schema (GUS) developed in the Computational Biology and Informatics Laboratory at the University of Pennsylvania (see: *http://www.gusdb.org*/). Thanks to its high-performance data access, complex query capabilities and easy Web site setup provided by the GUS Web Development Kit (WDK) (see: http://www.gusdb.org/wdk/), the GUS system with WDK has been adopted by many bioinformatics projects, including PlasmoDB, CryptoDB, ToxoDB, TcruziDB, GeneDB, BiowebDB, the Centromere Analysis System, the Phytophthora genome project, and others. These data resources are distributed over and linked by the Internet. Although they share the same database schema GUS, they use it in different ways, leaving integrators with manageable heterogeneities. Hence, the GUS databases have great potential to provide collaborative support for users who need to query across multiple databases.

The approach chosen for integration is determined by the project's requirements and objectives. The ApiDB project has the following objectives:

- Integrate three operational databases at two research institutions (CryptoDB at the University of Georgia, PlasmoDB and ToxoDB at the University of Pennsylvania).

- Provide biologists with uniform, integrated and centralized access to CryptoDB, PlasmoDB and ToxoDB.
- Minimize any interference with current operations or software infrastructure, that is, maintain local autonomy.
- Support local as well as global read access.
- Support local write access only via well-tested pipelines.
- Maintain and extend the easy-to-use interfaces to the databases provided by the GUS Web Development Kit (WDK).
- Provide rapid response times (a few seconds) for distributed queries. This requires efficient processing of distributed queries.
- Make sure the approach is scalable to allow other databases to join the integration without significant degradation of performance.

With all these requirements in mind, the ApiDB project is composed of two main modules separated by functionality, a front-end module that provides uniform Web access for biologists and a data store module that provides necessary data to the front-end module as depicted in Figure 2.1.

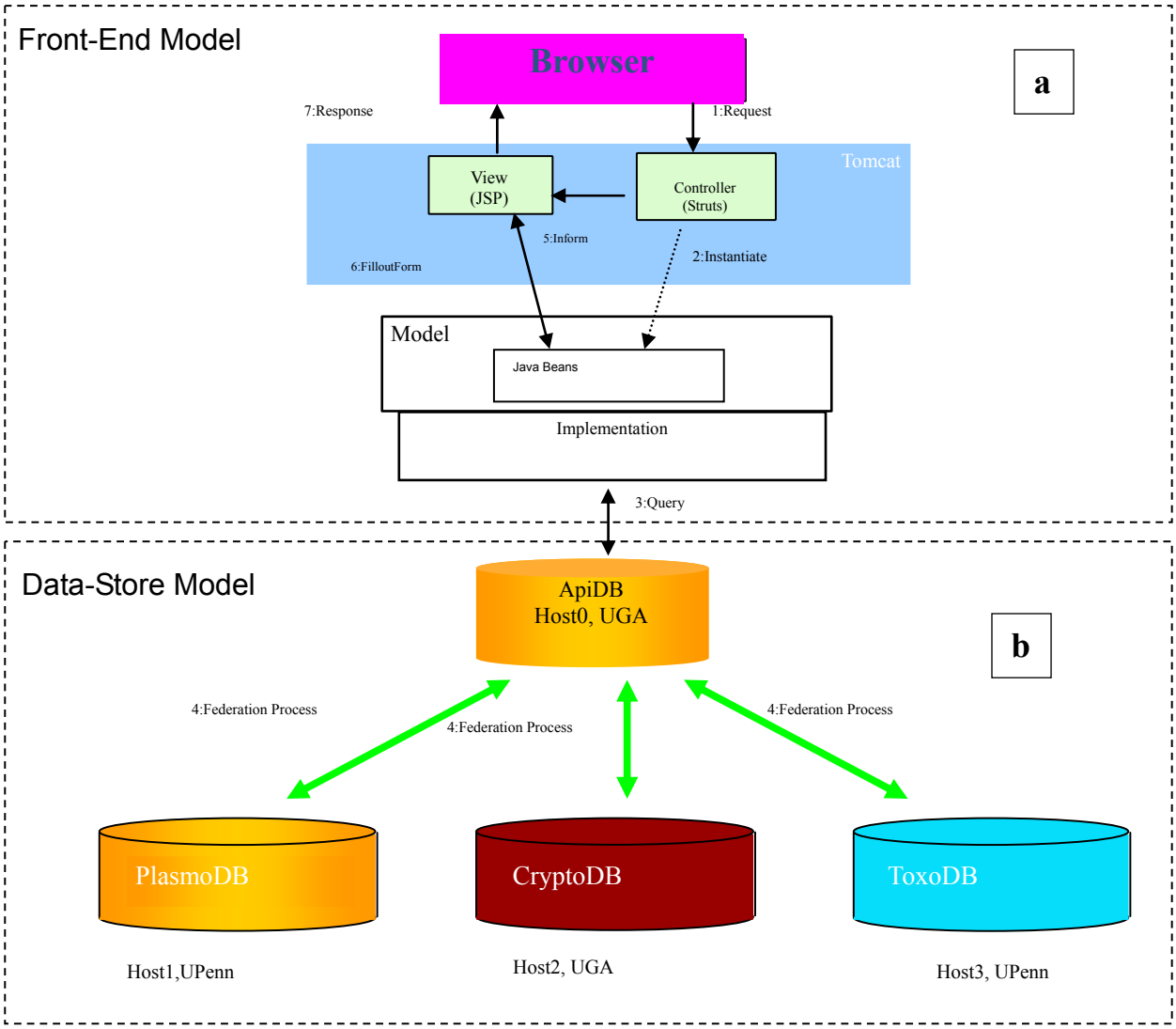

Figure 2.1. General ApiDB Architecture. The upper panel (a) illustrates the front-end module based on a Model-View-Controller (MVC) framework. The bottom (b) shows the data store module that integrates the three existing databases. The steps are numbered to represent the data flow whenever an end user queries the ApiDB website.

The ApiDB Web interface (front-end module, Figure 2.1) is created using the GUS Web Development Kit (WDK). The WDK is designed to significantly accelerate the development of Web sites and services that offer sophisticated querying facilities just like the EBI SRS server did

(Zdobnov et al., 2002). It can work on any relational database system and on any schema[5]. The WDK is a Java-based solution that uses a MVC design. Through an XML Model configuration file, the developer can define biological questions along with the corresponding SQL queries to answer the question. The WDK will decompose and translate those abstract SQL queries into real SQL queries on the fly, and then execute them to retrieve the results. The WDK can optimize performance by separating queries into two stages, first retrieving a list of primary keys, and then fetching extra attributes of a small chunk of data in response to a biologist's choice. It can also cache previous query results into primary key cache tables for future use to boost performance and to maintain a query history to be used by end users. Whenever a biologist wants to get the answer to a question, for example, searching a gene by id, the WDK will check if this question with the value of parameters has been asked before or not; if yes, the WDK will append the corresponding primary key cache table into the SQL query on the fly and skip the primary key SQL query to boost performance. If not, the WDK will create the primary key cache table according to the result of the primary key query with the value of parameters, and then append this primary key cache table to the SQL query on the fly. Figure 2.2 is the screenshot of the ApiDB website.

5. This feature greatly simplifies the schema mapping process in Federated Databases. It achieves schema mapping through mapping biological questions with corresponding SQL queries.

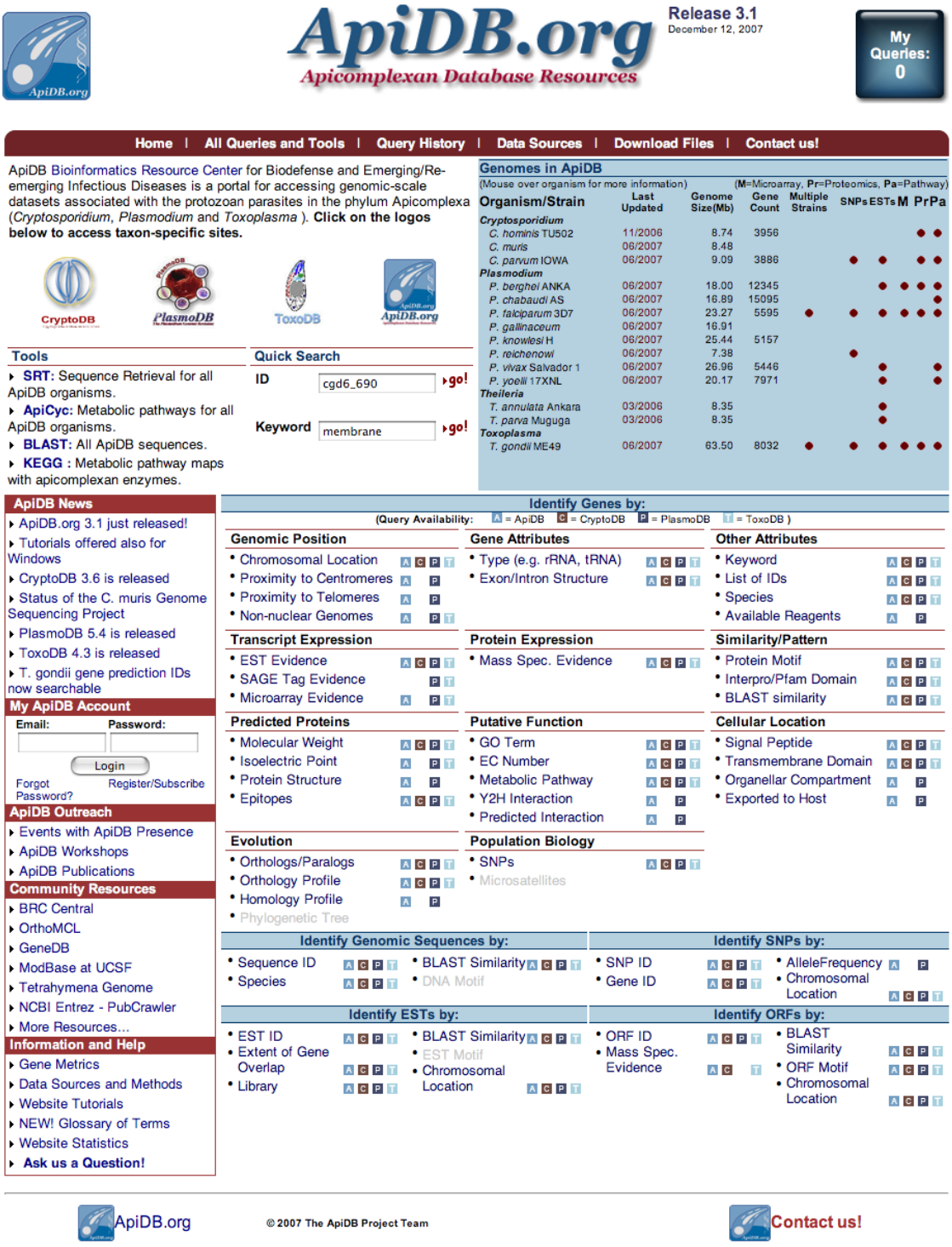

Figure 2.2. Screenshot of ApiDB (*http://apidb.org/apidb/*). Top right part is the general genomics information in ApiDB to show gene counts and data types for each organism. Bottom right part is the detailed questions that biologists can ask.

The databases in our project are distributed in that they are hosted by different organizations. One of the main aspects of Federated Databases is autonomy; that is, each component database is managed by its local organization, while at the same time it is participating in the federation. Autonomy also means that the federated database management system cannot completely control the component databases, which is true in our case. They are autonomous in that data is loaded and managed independently. There are many contributing factors to heterogeneity, for example, hardware, system software (such as operating systems) or database systems. A substantial amount of effort in heterogeneous Federated Databases is often needed for schema integration/matching. Though developed through a unified GUS schema, the component databases are heterogeneous to a certain degree as they use different DBMS's such as different versions of Oracle and PostgreSQL, and as GUS is a comprehensive schema with hundreds of tables, different labs have used it in variant ways. For example, the column "source_id" in the table "DoTS.TranslatedAASequence" can be gene's id or protein's id. Either way is functionally correct. Another major source of complexity in federated databases, global transactions, was not present in our case, since all updates occur locally to the component databases.

The mandate for the ApiDB project (Aurrecoechea, 2006) is to integrate multiple existing component databases, while maintaining local autonomy of each component. The Data Warehouse approach would have made maintaining local autonomy difficult. With the Federated Database approach (Figure 2.1), we need not copy the data from component databases as in a Data Warehouse. The federation approach provides us the most flexibility. In practice, we sometimes combine Federated Databases with Data Warehouses to boost performance through centralizing sequence data. In order to improve the performance for a single database, the WDK

will cache primary key tables in the component database. This causes a performance issue for ApiDB, which will be discussed later.

6. IMPLEMENTATOINS OF APPROACHES FOR THE APIDB PROJECT

The four approaches (Database links, JDBC, Web services and Java RMI) that were considered for the ApiDB project will be discussed in detail in the following sub-sections. Each approach contains multiple implementations. One particular ApiDB feature which is helpful for understanding the implementations is the two query phases, which is used to provide maximum flexibility and performance. The first query phase (primary key query) returns primary keys only, the second query phase (attribute query) returns all the results with the attributes that biologists choose in the first query phase.

6.1. IMPLEMENTATION USING DATABASE LINKS

Oracle provides a database link facility to connect and retrieve information from remote databases[6]. It allows multiple distributed databases to be treated as if they were a single, integrated database by creating DB links and by transparently maintaining a database session in the remote database on behalf of the local application request. For example, after creating a DB Link[7] to CRYPTO.APIDB.UGA.EDU and PLASMO.APIDB.UGA.EDU at ApiDB pointing to

6. Both IBM DB2 and Sybase SQL Server provide similar functions.
7. There are two ways to create private, public or global DB link:
   First, CREATE [PRIVATE/PUBLIC/GLOBAL] DATABASE LINK <link_name> CONNECT TO CURRENT_USER USING '<service_name>'; we can use this link to access database of service_name (which is defined in tnsnames.ora) from the current database, just as if CURRENT_USER access it.
   Second, we can define directly without defining service_name in tnsnames.ora as following: CREATE (*continued from the footnote in the previous page*) [PRIVATE/PUBLIC/GLOBAL] DATABASE LINK <link_name> CONNECT TO CURRENT_USER USING "(DESCRIPTION=(ADDRESS=(PROTOCOL= TCP)(HOST = host.name)(PORT = 1521))(CONNECT_DATA =(SERVER = DEDICATED)(SERVICE_NAME = service_name)))"

CryptoDB and PlasmoDB, the following SQL query will be sent to ApiDB to retrieve data from CryptoDB and PlasmoDB according to the primary key cache table (query_result_10) stored at ApiDB:

```
SELECT * FROM query_result_10, dots.rnatype@CRYPTO.APIDB.UGA.EDU u,
              dots.genefeature@CRYPTO.APIDB.UGA.EDU g, ...
WHERE  g.source_id = u.source_id(+)   AND g.source_id = query_result_10.source_id
              AND result_index_column >= 1
             AND result_index_column <= 20  AND query_result_10.project_id = 'cryptodb' ...
UNION ALL
SELECT * FROM query_result_10, dots.rnatype@PLASMO.APIDB.UGA.EDU u,
             dots.genefeature@PLASMO.APIDB.UGA.EDU g, ...
WHERE  g.source_id = u.source_id(+)   AND g.source_id = query_result_10.source_id AND
result_index_column >= 1
             AND result_index_column <= 20  AND query_result_10.project_id = 'plasmodb'...
```

The above example is part of the “Annotated Keyword” question in the ApiDB project, which is used to find genes in which the "Product Description" field matches what a biologist types. This query will combine the results returned from CryptoDB and PlasmoDB through “UNION ALL”. The corresponding original definition in the model file before WDK processing is as follows:

SELECT *...* ***FROM*** *dots.rnatype@CRYPTO.APIDB.UGA.EDU u,*

dots.genefeature@CRYPTO.APIDB.UGA.EDU g, ...

WHERE *g.source_id = u.source_id(+) ...*

UNION ALL

SELECT *...* ***FROM*** *dots.rnatype@PLASMO.APIDB.UGA.EDU u,*

dots.genefeature@PLASMO.APIDB.UGA.EDU g, ...

WHERE *g.source_id = u.source_id(+) ...*

Compared with the two SQL queries, we can see WDK appends the primary key cache table "query_result_10" (only created on first request) and the indexes of returned results to the query that will be executed in ApiDB.

6.1.1. Oracle Execution Plan

After a SQL query is parsed, there are three steps to optimize it in Oracle: transforming queries, estimating costs and generating plans (Oracle Database Performance Tuning Guide). Oracle uses the Cost-Based Optimizer (CBO) as the default optimizer. The Rule-Based Optimizer (RBO) is out-of-date because it can not use statistical information to generate an accurate execution plan. In a word, Oracle 10g can automatically tune SQL statements according to statistical information using its Cost-based Optimizer (CBO) (Kimberly, 2004). However, Oracle can not collect statistical information from remote Oracle databases through DB links, which causes a significant performance problem for our project and will be discussed below.

6.1.2. Optimizing Distributed Queries that use DB Links

In practice, we found the DB Link approach can have a performance problem, especially when the number of returned results is large. By tracing Oracle's execution plan, we found the poor performance to be due to large data transfers between the central database and the component databases. For example, when the number of returned results is 62,132, the SQL query takes more than two hours. The following two SQL queries clearly show the problem. The only difference between the two SQL queries is the location of the primary key cache table. In the first query, all the tables are on the remote site and accessed through DB link PLASMO.APIDB.UGA.EDU from ApiDB. In the second query, the primary key cache table is located in ApiDB and all the other tables are in remote site and accessed through the same DB link.

First SQL query:

```
SELECT result_index_column, gf.source_id, tn.name as genus_species,
        SUBSTR(tn.name, 1, 1) || '. ' || SUBSTR(tn.name, INSTR(tn.name, ' ', 1, 1) +1) as organism
FROM uga_fed.query_result_2@PLASMO.APIDB.UGA.EDU,
     dots.GeneFeature@PLASMO.APIDB.UGA.EDU gf,
     dots.NaSequence@PLASMO.APIDB.UGA.EDU s,
    sres.TaxonName@PLASMO.APIDB.UGA.EDU tn
WHERE gf.na_sequence_id = s.na_sequence_id AND s.taxon_id=tn.taxon_id
    AND tn.name_class = 'scientific name' AND gf.source_id=uga_fed.query_result_2.source_id
```

Second SQL query:

```
SELECT result_index_column, gf.source_id,tn.name as genus_species,
        SUBSTR(tn.name,1, 1) || '. ' || SUBSTR(tn.name,INSTR(tn.name, ' ', 1, 1) +1) as organism
FROM samwzm.query_result_2, dots.GeneFeature@PLASMO.APIDB.UGA.EDU gf,
     dots.NaSequence@PLASMO.APIDB.UGA.EDU s,
     sres.TaxonName@PLASMO.APIDB.UGA.EDU tn
WHERE gf.na_sequence_id = s.na_sequence_id AND s.taxon_id=tn.taxon_id
     AND tn.name_class = 'scientific name' AND gf.source_id=samwzm.query_result_2.source_id
```

The execution plan, statistical information and performance results of first SQL query are showed in Appendix A1.

The first SQL query just takes 0.28 seconds while the second one takes 6.75 seconds. Comparing the above two execution plans and statistics, especially from row 9 of the second query's execution plan, it shows clearly that a large amount of data is transferred from the remote database to the central database in the second SQL query, while no such data transfer occurs in the first query. This transfer of data is caused by Oracle doing the final table joins on the central database in the second query, which explains the difference in performance. In the first query, because all the table joins are completed in the component database, Oracle can use statistical information to generate a good execution plan and only transfer the final result to the central database.

We use two approaches to optimize distributed SQL queries: Oracle "hint" and decomposing the distributed SQL query, which will be discussed in the following subsections.

- ***DB Link with "hint"***

Using statistics, Oracle will generate the best execution plan to perform a SQL statement. Statistical information is critical for a good execution plan. Oracle usually can collect accurate statistical information for a single database, but it is not the case for remote database/distributed queries. This greatly affects ApiDB's performance because nearly all the queries in ApiDB are distributed queries through database links. We can use "hints" to let Oracle try to generate a better execution plan instead of using the default one, although it is not always the case. For example, the "DRIVING_SITE" hint is used to specify the database to handle the table join process. The default Oracle behavior is to join all the tables locally. In our case, that means the entire join will be processed in ApiDB which requires transferring a large amount of data from component databases to the central database in order to join with the primary key cache table which is small and stored in the central database. This will slow the performance considerably, which is the reason why the query takes more than two hours. Through "DRIVING_SITE", we can let Oracle send the primary key cache table to the component databases and join the tables at the component databases to significantly improve the performance. For example, the exact same SQL query as the previous one except using "DRIVING_SITE" improves performance as follows:

```
SELECT /*+DRIVING_SITE(gf) */ result_index_column, gf.source_id, tn.name as genus_species,
        SUBSTR(tn.name,1, 1) || '. ' || SUBSTR(tn.name,INSTR(tn.name, ' ', 1, 1) +1) as organism
FROM samwzm.query_result_2, dots.GeneFeature@PLASMO.APIDB.UGA.EDU gf,
     dots.NaSequence@PLASMO.APIDB.UGA.EDU s,
     sres.TaxonName@PLASMO.APIDB.UGA.EDU tn
WHERE gf.na_sequence_id = s.na_sequence_id AND s.taxon_id=tn.taxon_id
    AND tn.name_class = 'scientific name' AND gf.source_id=samwzm.query_result_2.source_id
```

Please refer to the execution plan, statistics information and performance result of this query in Appendix A2.

Compared with the former SQL, it speeds up from 6.75 seconds to 0.35 seconds. Unfortunately, the “DRIVING_SITE” hint does not work well for distributed queries containing more than two databases as in our situation. Once there is more than one "UNION ALL" in the SQL statement, which ApiDB requires because of queries to three component databases, Oracle ignores the “DRIVING_SITE” hint and uses the sub-optimized execution plan.

Trying to change Oracle's execution plan is very risky. For example, for a SQL query with three distributed component databases, the elapsed time is 16 seconds. While part of the previous SQL query with only two distributed component databases costs more than 50 seconds! Apparently Oracle generates a sub-optimized execution plan in the later case. Even if we could find a way to make Oracle use the optimized execution plan, it is not stable and not transportable and is a burden to model developers. In short, there are many things that can affect Oracle's execution plan and there is no way to guarantee that Oracle will generate an optimal execution plan and abide by it for the distributed query.

From these observations, a Pure Oracle Execution Plan seems to not be a good choice for ApiDB. However, we can combine the hint of “DRIVING_SITE” with other approaches to take advantage of Oracle's optimizer.

- ***Decompose SQL query combined with “hint”***

First, according to the position of “UNION ALL”, decompose the single complex SQL query into three component SQL queries such that each component query only contains one kind of DB Link. Second, add a "DRIVING_SITE" hint in each of component SQL query in order to

let Oracle generate an optimized SQL execution plan. Third, run the component SQL queries at ApiDB. Fourth, wrap the three ResultSets returned from three component SQL queries into a CachedRowSet in memory and upcast it to ResultSet. Both ResultSet and CachedRowSet are the data types used to represent query results. Both of them provide methods to retrieve and update data from the database. Unlike ResultSet that will not contain any data itself and keeps a connection with database all the time, CachedRowSet will disconnect with database once it is finished populating itself with all the data. That is the reason why CachedRowSet will use more memory than ResultSet, which is a disadvantage of this approach. The advantage of this approach is the great performance improvement it can provide. Besides, CachedRowSet can handle more client requests at one time than the ResultSet approach because CachedRowSet will not occupy any resources from database.

As the third step, we test two implementations: serial querying and parallel querying using threads. Unlike serial querying, parallel querying uses threads to query the component database simultaneously. Theoretically, the overall time is the longest component SQL statement time, which can improve the performance, while increasing memory usage due to the threads. Please refer to the detailed evaluation in section 8, “Performance Evaluation and Discussion”.

6.2. IMPLEMENTATION USING JAVA DATABASE CONNECTIVITY

Java Database Connectivity (JDBC) technology provides a convenient way to embed SQL statements in applications to communicate with databases. It is more generic compared to Oracle DB Link, since it can connect with any kind of databases as long as there is suitable database driver. However, Oracle JDBC usually needs port 1521 open to access databases behind firewalls, which has potential security problem, just like Oracle DB Link strategy. In practice,

port 1521 of component's database host only opens to the host of ApiDB database, which greatly reduces the potential security problems.

One of the main issues with this approach is how to deal with primary key cache tables. By default, the primary key cache tables are generated and stored in ApiDB, because of the default behavior of the WDK. According to the location of the primary key cache tables, we have three implementation options:

- Copying cache tables from ApiDB to component databases.

  Before every distributed query, the WDK needs to copy the corresponding primary key cache table from ApiDB to the component databases. Then, instead of connecting with ApiDB, the WDK will connect with component databases directly through JDBC and retrieve the results in CachedRowSet format and finally merge the component results in memory. Since this implementation requires copying the cache table before every distributed query, it will affect the performance. In practice, the primary key cache table is copied from ApiDB to the component databases whenever it is created in ApiDB.

- Creating cache tables on both ApiDB and component databases.

  Whenever creating or updating primary key cache tables on ApiDB, the WDK will make a clone on the component databases. Since the primary key cache table on component databases always keeps synchronization with ApiDB, there is no need to copy the cache table from ApiDB to component databases as in the previous implementation. The main problem with this approach is that we must modify WDK's cache table generation code that is very complex and has already been validated at the component databases.

- Connect to component databases directly and join cache table remotely.

  In this approach, cache table creation need not be changed at all. Whenever there is

distributed query, the WDK will connect with component databases directly through JDBC and join cache table located on ApiDB through DB Link at component databases pointing to ApiDB. Since the primary key cache tables are usually small compared with tables on component databases, Oracle 10g can generate optimized execution plans based on its default behavior. The main problem of this approach is that we need to dynamically modify SQL query statement since WDK will append the information about the primary key cache table into SQL statement on the fly.

The first approach is implemented as our JDBC federation approach. That is, copying the cache table from ApiDB to the component databases and then connecting to component databases directly through JDBC. The following is the JDBC architecture.

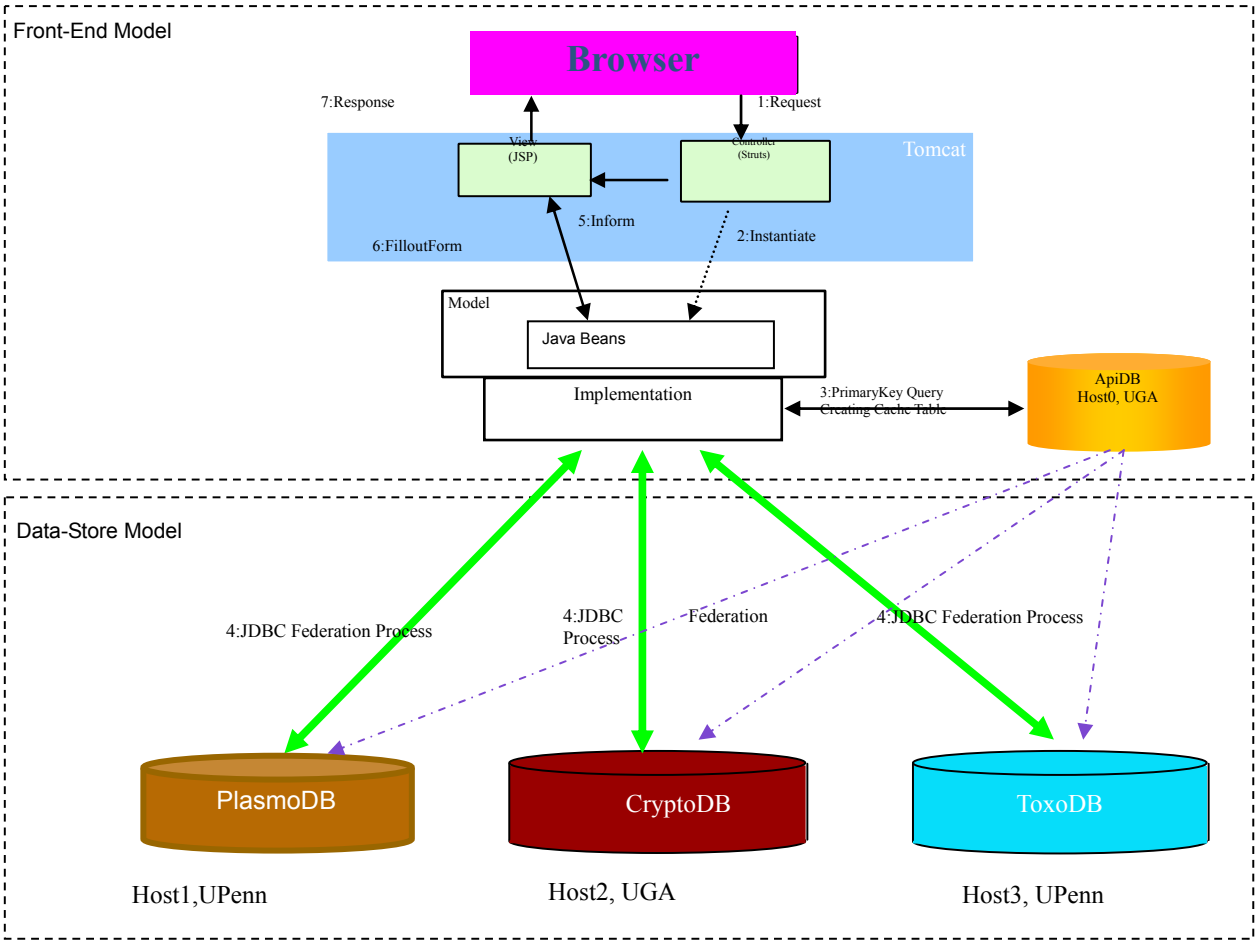

Figure 2.3. Architecture of JDBC Approach**.** Similar to Figure 2.1, except that federation part is achieved by JDBC technology. Also, the central ApiDB database is used to store a primary key cache table.

When retrieving a large number of rows from remote databases through JDBC, setting ResultSet's fetch size can boost performance dramatically. Fetch size controls how many rows to be retrieved from a database at one time. Network delay is a very important factor affecting performance, which is our case because ApiDB is located at the University of Georgia, while PlasmoDB and ToxoDB component databases are located at the University of Pennsylvania. The default fetch size is 10, which is the setup for JDBC implementation one (JDBC1). In JDBC

implementation two (JDBC2), the ResultSet's fetch size is adjusted to 6000. The disadvantage of a large fetch size is that it takes more memory, even exceeding Java's heap memory causing the program to exit abnormally if it is made too large.

Another problem is that some methods of ResultSet cannot be used safely in CachedRowSet, which is a Java design problem. According to the Liskov Substitution Principle (LSP), all the methods of parent class should be applicable to a child class unless the child class re-implements the method. Although CachedRowSet is inherited from ResultSet, not all ResultSet's methods are applicable to CachedRowSet. The possible reason is the CachedRowSet's connection to database is closed, not like ResultSet that holds the connection to database all the time. For example, ResultSet's findColumn (attributeName) and getObject (attributeName) methods can not be used in CachedRowSet. In both cases, attributeName needs to be replaced with the corresponding columnIndex that can be obtained from ResultSet's getMetaData method.

6.3. IMPLEMENTATION USING WEB SERVICES

Web Services could be the most widely applicable approach to ApiDB among all the other integration approaches. There is no direct connection between the central database and the component databases. They are not aware of the existence of each other outside of the communication between amongst the Web Services. This decoupling brings many benefits. For example, both client and server can change platform, database schema, or even switch to another DBMS. Changes in class definition or coding language can be made without any ill effects. Both client and server can add more functionality with greater simplicity. Suppose a new BRC resource is brought online with Web services having a mappable interface, it can be incorporated

into the federated system with minimal effort.

In the ApiDB project, there is a Web service for each component database to provide data query services. Whenever there is a distributed SQL query in ApiDB, it will first be decomposed into component SQL queries and then wrapped into a Web service request and sent to the component Web service. Each remote Web service communicates with its local component database to retrieve data according to wrapped SQL queries.

We choose Axis2 1.0 as the Web Services platform, an open source project provided by Apache (see: http://ws.apache.org/axis2/). It is a new design with many new features. For example, with its own object model called AXIOM and StAX (Streaming API for XML) parsing, Axis2 is much faster than earlier versions of Apache Axis. According to the study of Srinivas (http://www.wso2.net/2006/05/axis2_performance_testing_round_1), Axis2 is 4~5 times faster than Axis1. AXIOM stands for AXis Object Model representing XML infoset model, which has good extensibility, high performance and flexibility. Axis2 uses HTTP 1.1 and supports both SOAP 1.1/SOAP 1.2 and the emerging Representational State Transfer (REST) style of Web services. Furthermore, it provides binary attachments and asynchronous Web service invocation, which means the client, does not need to block at the invocation point while waiting for the response from the server side. The client can continue working after firing invocation to the server; the server will call back after finishing the invocation. Asynchronous Web Services invocation is very useful if the server needs a long time to finish the invocation.

- XML data exchange format

  After receiving SQL query from a central WDK invocation, the Web service populates ResultSet into WebRowSet and serializes to XML format which is transferred back to the central site in asynchronous invocation style, which will avoid disconnection caused by long

time invocation. This may happen in our case because some SQL queries will need a much longer time to finish than the default Web Service connection time. After getting WebRowSet from the Web service, the central site will merge them into a single WebRowSet object in memory. In this approach, since the results are encoded into XML format, they can be processed by any XML aware application.

- Binary attachment

  After receiving an SQL query from the central site invocation, the Web service populates CachedRowSet into ResultSet and transfers it back as a binary attachment. As with the former implementation, after getting the CachedRowSet from Web service, the central site will merge them into a single CachedRowSet object in memory.

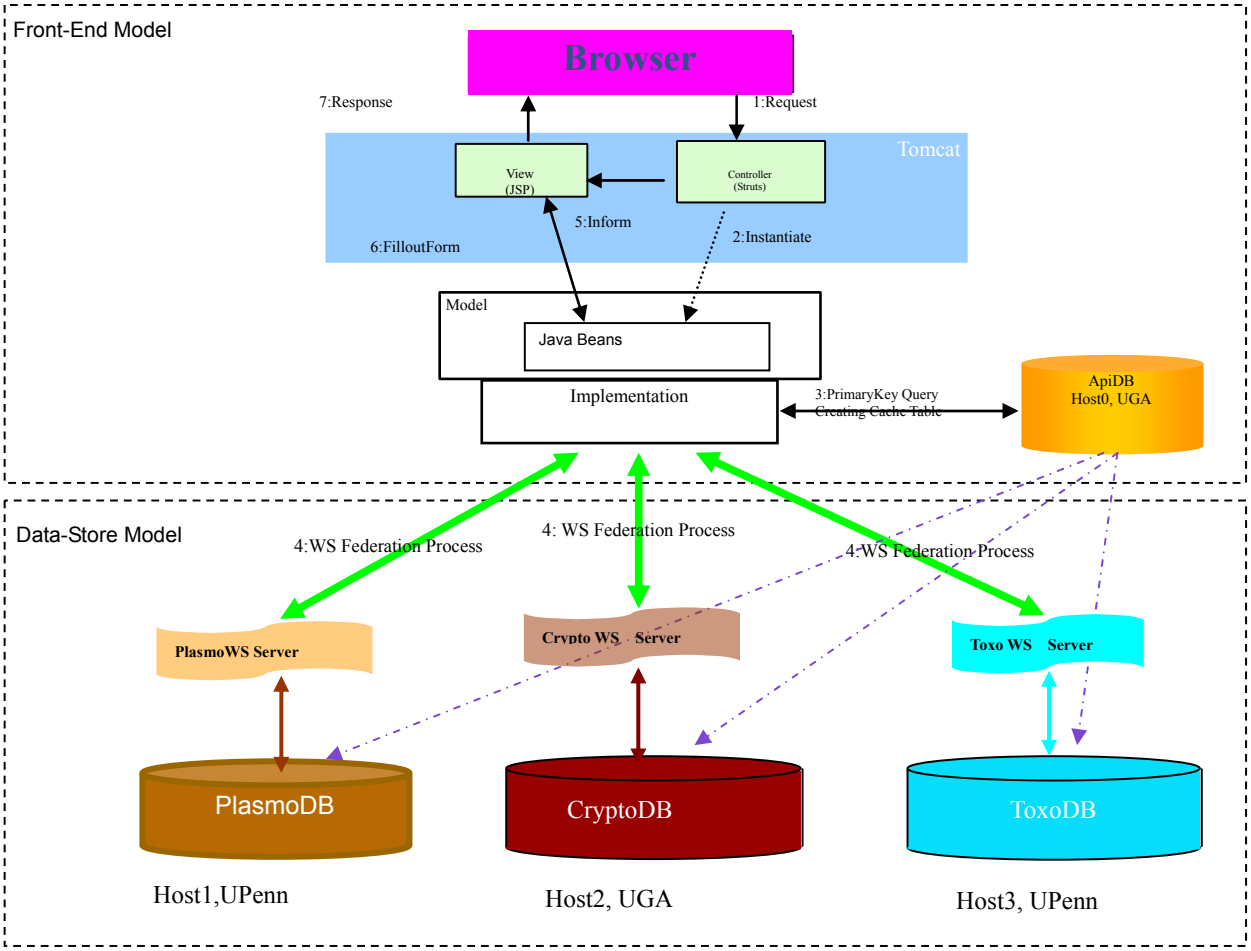

Figure 2.4. Architecture of Web services Approach**.** Similar to Figure 2.1, except that the WDK communicates with the web servers for each component database. Also the central ApiDB database stores a primary key cache table.

6.4. IMPLEMENTATION USING JAVA RMI

The Java Remote Method Invocation (RMI) approach is similar to the Web services approach but includes some restrictions (*e.g.* Java language, more primitive forms of discovery, *etc*. (Sheth and Miller, 2003)). A SQL query is first decomposed to component queries and rewritten according to the component database schema where each primary key cache table is

located. Then three RMI invocations from the central site to the corresponding remote RMI Server occur simultaneously with the corresponding component SQL query as the parameter. The returned result is populated into CachedRowSet. After all the CachedRowSets from CryptoDB, PlasmoDB and ToxoDB have been returned to ApiDB, they will be merged into one CachedRowSet in memory. Figure 2.4 is the architecture of Java RMI implementation in the ApiDB project.

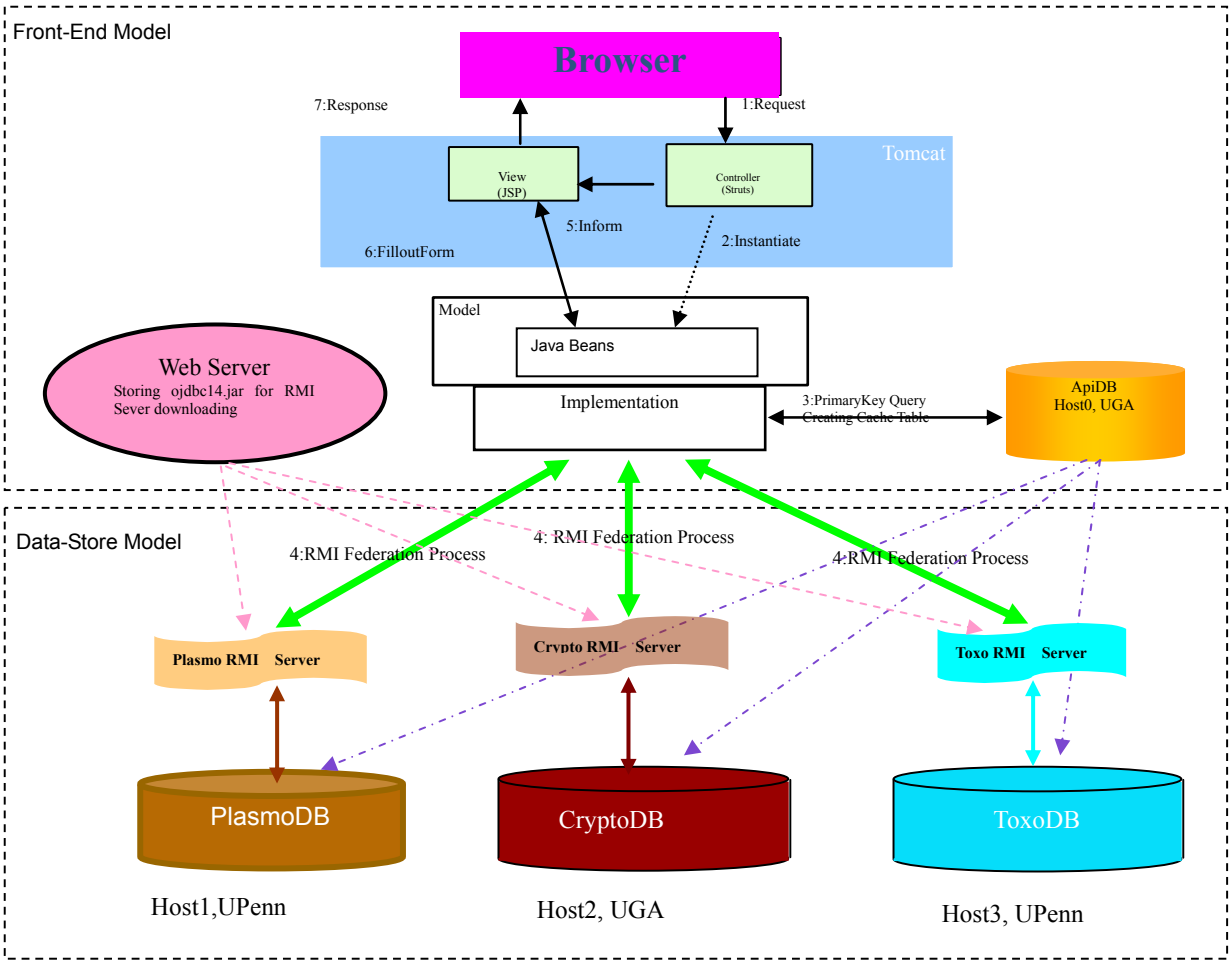

Figure 2.5. Architecture of JAVA RMI Approach**.** Similar to Figure 2.4, there is a Java RMI server for each component database, providing data to the central ApiDB. Furthermore, another central Web Server provides public resources, such as Oracle's JDBC driver, to all the RMI servers.

7. PERFORMANCE EVALUATION AND DISCUSSION

There is no doubt that the system and architectural design are the most important issues for the success of any software system. At the same time, performance is another important concern. The system will become useless if the general performance is too slow to meet users' needs. Table 2.1 lists the eight different implementations that are evaluated for the ApiDB project[8]:

Table2.1. Implementations for ApiDB Federation

NAME	APPROACH	IMPLEMENTATION DESCRPTION
CachedRowSet	DB Link	Decompose SQL query combined with Oracle "hint" that executes component SQL queries serially. A dark blue bar in Fig. 2.6, 2.7 and 2.8. (#1)
Multi CachedRowSet	DB Link	Decompose SQL query combined with Oracle "hint" that executes component SQL queries parallel. A red bar in Fig. 2.6, 2.7 & 2.8. (#2)
JDBC1	JDBC	Decompose SQL query in ApiDB and connect to component site directly with JDBC, setting fetch size of ResultSet as 10. A yellow bar in Fig. 2.6, 2.7 & 2.8. (#3)
JDBC2	JDBC	Decompose SQL query in ApiDB and connect to component site directly with JDBC, setting fetch size of ResultSet as 6000. A green bar in Fig. 2.6, 2.7 & 2.8. (#4)

8. Hardware specification of the central site: Intel® Xeon™ CPU 2.40GHz; 1GB RAM, Red Hat Enterprise Linux WS release 3 (Taroon Update 7). Gigabit Internet Connection. Oracle (Enterprise 10.0G) Database Server (shared by many other applications): Sun E6500, 14CPU's and 20GB of RAM. Gigabit Internet Connection.

RMI	Java RMI	Decompose SQL query in ApiDB and communicate with remote sites through Java RMI technology. A purple bar in Fig. 2.6, 2.7 & 2.8. (#5)
WS1	Web Service	Decompose SQL query in ApiDB and communicate with remote sites through Web Service, the result (WebRowSet) is encoded in plain XML format, An orange bar Fig. 2.6, 2.7 & 2.8. (#6)
WS2	Web Service	Decompose SQL query in ApiDB and communicate with remote sites through Web Service, the result (CachedRowSet) is transferred as binary attachment. A brown bar Fig. 2.6, 2.7 & 2.8. (#7)
ResultSet	DB Link	Original approach using DB Link without decomposing the SQL query or combining "hint". A grey bar in Fig. 2.6 & 2.8. (#8)

In order to compare the performance among the eight implementations, a shell script was made to save the output and measure the query time. It calls a class in WDK to retrieve the answer to a question defined in the XML model file and print the result out to the screen. We use the Linux "top" application to measure memory usage, which is a part of procps package. It measures the overall memory usage including the JVM, pre-allocated memory, stack memory and heap memory. The most time-consuming question among all of the ApiDB's questions—the "Annotated Keyword" question was chosen for evaluation. To test the scalability of performance, seven different parameters (phosphoenolpyruvate, da*, de*, ed*, e*, hypo*, *) were chosen for the "Annotated Keyword" question in order to return different result sizes (18, 598, 2234, 10508,

17191, 42905, 62443 rows, respectively). A detailed time comparison and analysis will be given in section 7.1, following memory comparison and analysis in section 7.2.

7.1. TIME ANALYSIS

The detailed results of the timing analysis are shown in Figures 2.6 and 2.7. The only difference between these two figures is that Figure 2.7 removes the original Oracle DB Link approach due to its poor performance. It is so slow that some of the data from the other approaches is not visible in Figure 2.6.

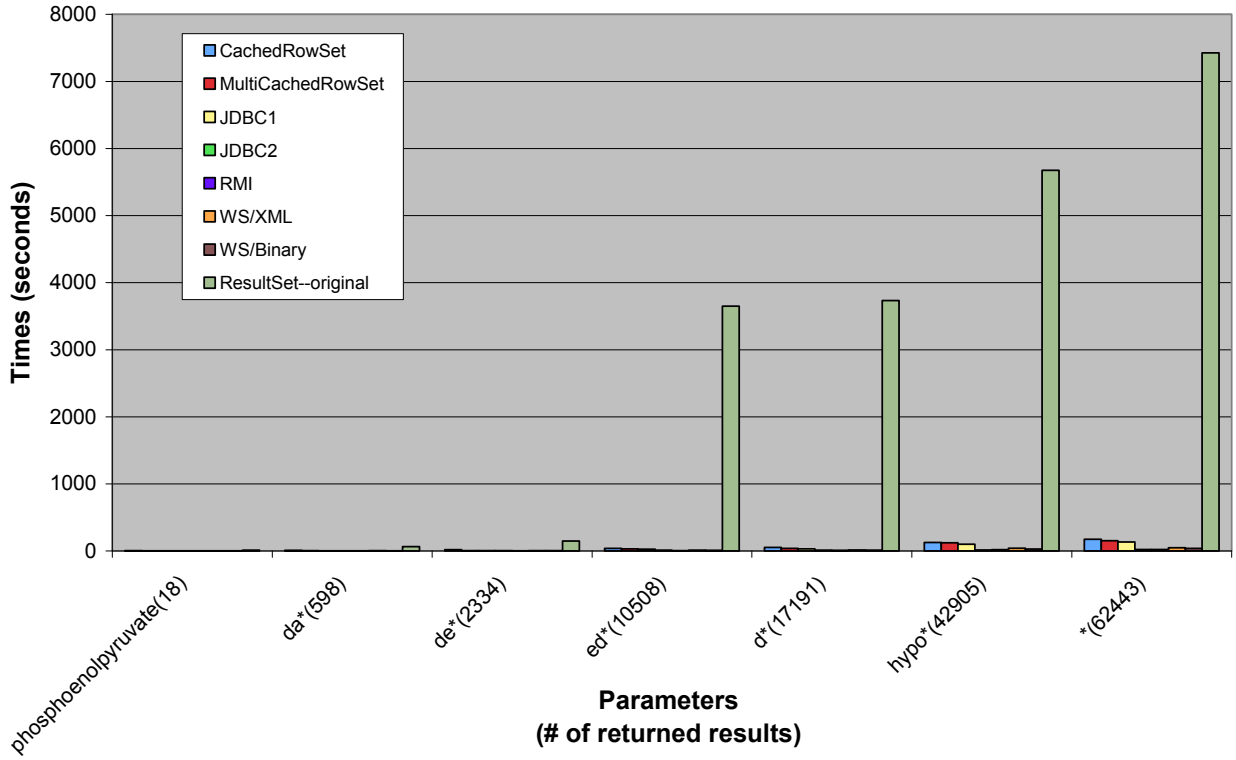

Figure 2.6. Overall Timing Comparison.

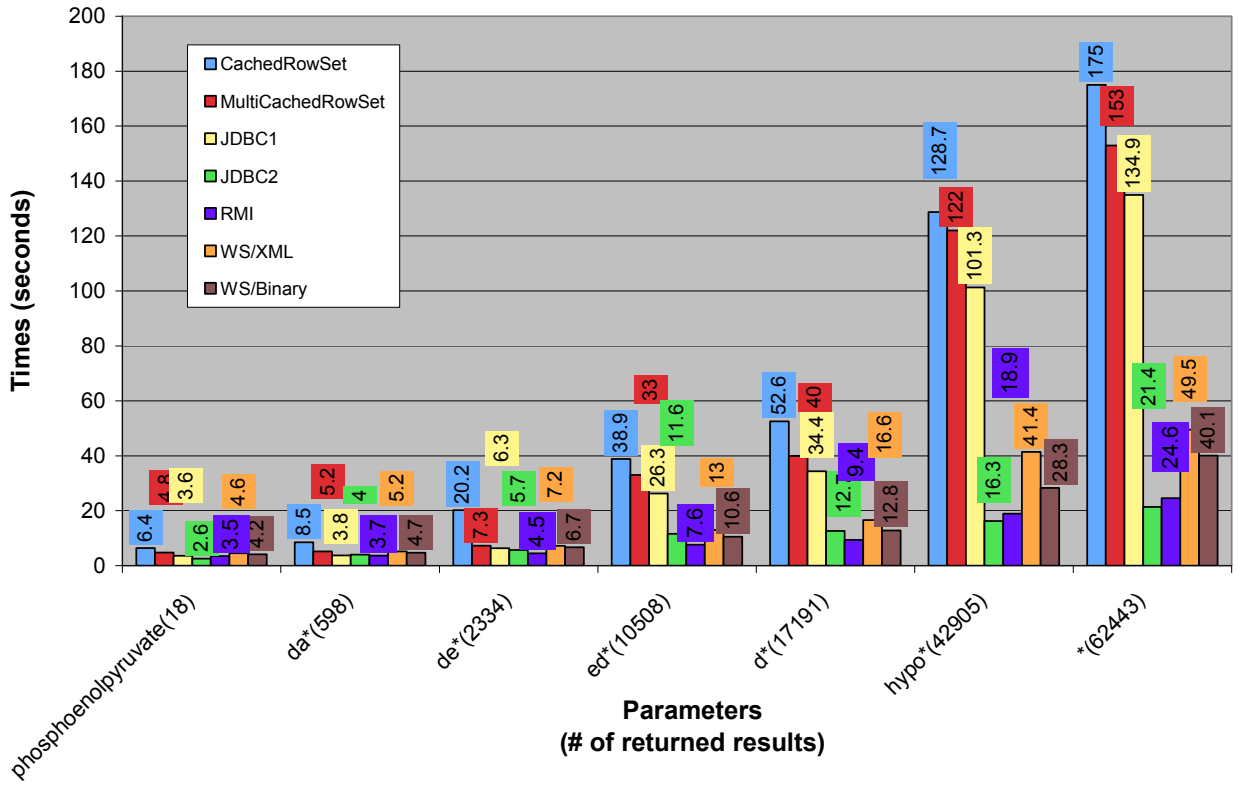

Figure 2.7. Timing Comparison without the original federation approach.

From Figure 2.6, we can see the original Oracle DB Link implementation is much slower than all the other implementations, especially when the number of returned results is very large. For example, when the number of returned rows is 62,443, it takes more than two hours to finish answering the question, which is unacceptable. This situation arises because Oracle can not generate optimal execution plans; the sub-optimized execution plan causes a large transfer of data between the central database and the component databases via the network which is the main reason for the inefficiency. Compared with the original Oracle DB Link implementation, the other two Oracle DB Link implementations only transfer a very small amount of data through the network. From Figure 2.7, we can see JDBC2 and RMI are the fastest implementations.

When the parameters are da*(598), de*(2334), ed*(10508) and d*(17191), Java RMI is the fastest implementation. JDBC2 is the fastest implementation on the other parameters. All of the first seven implementations are much better than the original. For example, when the parameter is *(62443), the JDBC2 implementation is 353 times faster than the original implementation and Java RMI is 309 times faster than the original implementation. JDBC1 is slower than JDBC2, especially when the number of returned results is large because of network delay. JDBC1 needs much more time to retrieve all the data from the remote databases compared with JDBC2. The most interesting discovery is that Web service technology is very competitive. When the number of returned results is less than 17,000, the Web Service implementation is nearly the same as JDBC2 and better than JDBC1. Part of the reason is Web Services return all of the results at once instead of returning multiple times like JDBC as it iteratively transfers the ResultSet. With increasing numbers of returned results, Web Services become slower than JDBC2 because Axis2 needs more time to prepare the result to transfer and convert back after receiving the result. WS2 is always faster than WS1 because there is no need to serialize WebRowSet to XML and vice versa, also the size of XML is larger than the size of the binary attachment.

We can see that the multi-threaded JDBC implementation is better than serial JDBC implementation (because we send the component queries to ApiDB in parallel), especially when the number of returned results is less than 10,000. For example, when the number of returned result is 2272, the multi-threaded version is nearly three times faster than the serial version. With the increase of the number of results returned, the benefits of the multi-threaded version decreases for two reasons: First, the returned result is unbalanced because PlamsoDB contains much more data than CryptoDB and ToxoDB. Second, most of the time is wasted on waiting for data transferred through network which is more serious for our situation, because the University

of Pennsylvania which hosts PlasmoDB and ToxoDB databases is far away from the University of Georgia which hosts CryptoDB and ApiDB databases.

7.2. MEMORY USAGE ANALYSIS

The summary results for memory usage analysis are shown in Figure 2.8.

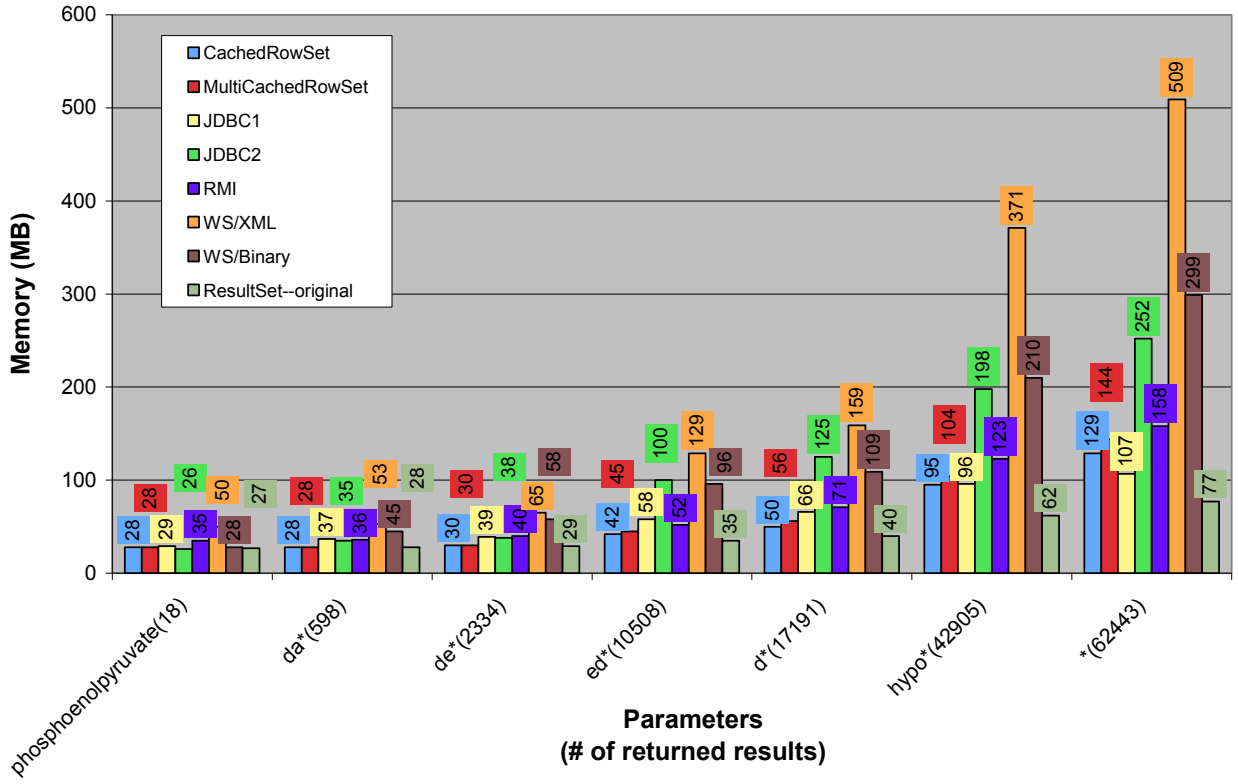

Figure 2.8. Memory Usage Comparison.

From Figure 2.8, we can see the original approach which uses ResultSet to represent the result always consumes the least amount of memory in all sitations compared with the other implementations. The reason for this is that ResultSet is a kind of iterator which only provides a

hook to the caller. All the other implementations use CachedRowSet/WebRowSet to read all the data from the databases into memory, so they are expected to use more memory than ResultSet. In all cases, Java RMI uses less memory than JDBC2, while more than JDBC1. WS1 consumes the most memory because of character encoding in the XML format. Also, we can see JDBC2 uses more memory than JDBC1 because of the higher fetch size value. Considering it is very rare that the end user wants to return more than 10,000 results at one time, all the implementations are likely to meet ApiDB's memory capacity limitations.

8. DISCUSSION AND CONCLUSIONS

With the explosive increase in the quantity and scope of biological data, the need for effective and practical data integration of bioinformatics data is ever increasing. In this paper, we have outlined several feasible approaches for data integration for the ApiDB project as an example to illustrate how to federate biological databases. The goal of the ApiDB project is to provide biologists with uniform, integrated and centralized Web access to apicomplexan genome resources by integrating multiple distinct biological databases, hosted by the University of Georgia and the University of Pennsylvania. We believe that providing centralized and uniform access to integrated information will greatly help biologists to obtain a deeper understanding of the fundamental biology of sets of pathogenic organisms in order to counter the threats posed by these pathogens.

When faced with a data integration problem, there are several factors to weigh when choosing an approach. The factors of main concern to us in the ApiDB project were to provide transparency to end users, ease of implementation/extension and good performance. In addition, it was desirable to maintain a level of local autonomy. The approaches we have considered are

the following: Database Links, Java Database Connectivity (JDBC), Web services and Java Remote Method Invocation (Java RMI). Note that the first two approaches provide different ways of achieving a Database Federation, while the latter two may be considered as middleware approaches. We further evaluated multiple particular implementations of these approaches to quantitatively evaluate their performance. This paper also presents our rationale for not including other possible approaches such as Link Integration, Query-Based Integration (see section 2) or Data Warehouses (see section 3) in our evaluation.

Traditionally, integration has been at the database (or bottom) layer, while more recently integration has started to occur higher up in the middleware. Our investigation includes both layers. Although it is difficult to declare a winning approach, we feel that the JDBC based Database Federation approach is likely the best at the bottom layer, especially when maintaining local autonomy is an important factor. Compared with other common database integration approaches, such Database Federation approaches can minimize the disruption of current operations, maintain local autonomy, and handle heterogeneities as well as scalability, the most important concerns in database integration. However, the Database Federation approaches are not easy to extend compared with the Web services approach. Because of platform independence, language independence and decoupling of service from client, the Web services approach provides the most flexibility, especially when the system needs to be extended to incorporate more resources. From our performance evaluation of response time for the eight different implementations, the top three, JDBC with a large fetch size, Java RMI and Web services, differ by less than a factor of two, while the bottom three, Database Links, multithreaded Database Links and JDBC with a small fetch size, are much slower (up to an order of magnitude slower). Although the Java RMI approach performs better than the Web services approach, it has

more restrictions. Considering the flexibility and competitive performance that the Web service approach can provide, it has the potential to provide an excellent solution for biological data integration problems. Furthermore, based on experience from the ApiDB project, Web services technology can meet the requirements for practical industrial strength data integration systems. At the same time, the move from integrating at the database level to the Web services level, allows new databases/organisms to be added to the federation much more quickly.

ACKNOWLEDGEMENTS

Z.W. was supported by NIH R01 AI058515 to J.C.K. We would like to thank all the ApiDB project team members for their help in this study. This project has been funded in whole or in part with Federal funds from the National Institute of Allergy and Infectious Diseases, National Institutes of Health, Department of Health and Human Services, under Contract No. HHSN266200400037C.

REFERENCES

SHETH, A.P & MILLER, J.A. (2003). WEB SERVICES: TECHNICAL EVOLUTION YET PRACTICAL REVOLUTION? IEEE INTELLIGENT SYSTEMS., 18(1), 78-80.

BATINI, C, LENZERINI, M & NAVATHE, S.B. (1986). A COMPARATIVE ANALYSIS OF METHODOLOGIES FOR DATABASE SCHEMA INTEGRATION. ACM COMPUT SURV 18(4):323-364.

CASEY, R.M. (2006). HOW FEDERATED DATABASES BENEFIT BIOINFORMATICS RESEARCH BUSINESS INTELLIGENCE NETWORK: THE VISION FOR BI AND BEYOND

CHEN, I.A. & MARKOWITZ, V.M. (1995). AN OVERVIEW OF THE OBJECT-PROTOCOL MODEL (OPM) AND OPM DATA MANAGEMENT TOOLS. INFORM.SYS., 20, 393–418.

DAVIDSON, S.B., OVERTON, C & TANNEN, V.B. (1997). A DIGITAL LIBRARY FOR BIOMEDICAL RESEARCHERS. INT J DIGIT LIBR.,1,1.

DAVIDSON, S. B. ET AL. (2001). K2/KLEISLI AND GUS: EXPERIMENTS IN INTEGRATED ACCESS TO GENOMIC DATA SOURCES. IBM SYST. J. 40.

DOAN, A.H., DOMINGOS, P & HALEVY, A. (2001). RECONCILING SCHEMAS OF DISPARATE DATA SOURCES: A MACHINE-LEARNING APPROACH. IN: PROC ACM SIGMOD CONF, PP. 509-520.

ELMAGARMID, A.K. & PU, C. (1990). GUEST EDITORS' INTRODUCTION TO THE SPECIAL ISSUE ON HETEROGENEOUS DATABASES. ACM COMPUT SURV 22(3):175-178.

ETZOLD, T. & ARGOS, P. (1993). SRS—AN INDEXING AND RETRIEVAL TOOL FOR FLAT FILE DATA LIBRARIES. COMPUT. APPL. BIOSCI., 9, 49–57.

TYSON, F. (2005). COMPARATIVE MOUSE GENOMICS CENTERS CONSORTIUM (CMGCC): MOUSE MODELS TO IMPROVE UNDERSTANDING OF THE BIOLOGICAL SIGNIFICANCE OF HUMAN POLYMORPHISMS. ENVIRON HEALTH PERSPECT 113(8): A545.

GALPERIN, M.Y. (2008) THE MOLECULAR BIOLOGY DATABASE COLLECTION: 2008 UPDATE NUCLEIC ACIDS RES., JANUARY 2008; 36: D2 - D4.

HAAS, L.M., SCHWARZ, P.M., KODALI, P., KOTLAR, E., RICE, J.E. & SWOPE, W.C. (2001). DISCOVERYLINK: A SYSTEM FOR INTEGRATED ACCESS TO LIFE SCIENCES DATA SOURCES. IBM SYS. J., 40, 489–511.

GUNTHER, H. (2003). PERFORMANCE BEST PRACTICES FOR USING WAS WEB SERVICES. WEBSPHERE DEVELOPER'S JOURNAL., VOL: 2 ISS: 10.

HUANG, H., BARKER1, W.C., CHEN1, Y & WU, C.H. (2003). IPROCLASS: AN INTEGRATED DATABASE OF PROTEIN FAMILY, FUNCTION AND STRUCTURE INFORMATION. NUCLEIC ACIDS RES. 31(1), 390-392.

KARASAVVAS, K.A., BALDOCK, R. & BURGER, A. (2004). BIOINFORMATICS INTEGRATION AND AGENT TECHNOLOGY. J. BIOMED. INFORM., 37, 205–219.

KARP,P. (1995). A STRATEGY FOR DATABASE INTEROPERATION. J. COMPUT.BIOL., 2, 573–586.

KEMP, G.J.L., ANGELOPOULOS, N & GRAY, P.M.D. (2002). ARCHITECTURE OF A MEDIATOR FOR A BIOINFORMATICS DATABASE FEDERATION. IEEE TRANS INF TECHNOL BIOMED 6,2.

KOHLHOFF, C. & STEELE, R. (2003). SOAP FOR HIGH PERFORMANCE BUSINESS APPLICATIONS: REAL-TIME TRADING SYSTEMS. IN WORLD WIDE WEB CONFERENCE.

GOTTSCHALK, K., GRAHAM, S., KREGER, H. & SNELL, J. (2002). INTRODUCTION TO WEB SERVICES ARCHITECTURE IBM SYS. J. 41.

LARSON, J.A., NAVATHE, S.B. & ELMASRI, R. (1989). A THEORY OF ATTRIBUTE EQUIVALENCE IN DATABASES WITH APPLICATION TO SCHEMA INTEGRATION. IEEETRANS SOFTWARE ENG 16(4):449-463.

WILKINSON, M.D. & LINKS, M. (2002). BIOMOBY: AN OPEN SOURCE BIOLOGICAL WEB SERVICES PROPOSAL. BRIEFINGS IN BIOINFORMATICS 3(4),331-341.

NAGARAJAN, M., VERMA, K., SHETH, A.P., MILLER, J.A. & LATHEM, J. (2006) "SEMANTIC INTEROPERABILITY OF WEB SERVICES - CHALLENGES AND EXPERIENCES," PROCEEDINGS OF THE 4TH IEEE INTERNATIONAL CONFERENCE ON WEB SERVICES (ICWS'06), CHICAGO, ILLINOIS (SEPTEMBER 2006) PP. 273-380.

MARTONE, M.E., GUPTA, A., WONG, M., QIAN, X., SOSINSKY, G., LUDAESCHER, B. & ELLISMAN, M.H. (2002). A CELL-CENTERED DATABASE FOR ELECTRON TOMOGRAPHIC DATA, J. STRUCT. BIOL. 138 (1-2), 145-155.

CORNELL, M., PATON, N.W., HEDELER, C., KIRBY, P., DELNERI, D., HAYES, A., OLIVER, S.G. (2003). GIMS: AN INTEGRATED DATA STORAGE AND ANALYSIS ENVIRONMENT FOR GENOMIC AND FUNCTIONAL DATA. YEAST. 20(15): 1291 – 1306

MIYAZAKI, S., SUGAWARA, H., GOJOBORI, T. & TATENO, Y. 2003. DNA DATA BANK OF JAPAN (DDBJ) IN XML. NUCLEIC ACIDS RES., 31, 13–16.

PARENT, C. & SPACCAPIETRA, S. (1998). ISSUES AND APPROACHES OF DATABASE INTEGRATION. CACM 41(5):166-178.

BUNEMAN, P., LIBKIN, L., SUCIU, D., TANNEN, V. & WONG, L. (1994). COMPREHENSION SYNTAX, ACM SIGMOD RECORD 23, 87-96.

PHILIPPI, S. (2004). LIGHT-WEIGHT INTEGRATION OF MOLECULAR BIOLOGICAL DATABASES. BIOINFORMATICS 20, 51-57.

RITTER, O., KOCAB, P., SENGER, M., WOLF, D. & SUHAI, S. (1994). PROTOTYPE IMPLEMENTATION OF THE INTEGRATED GENOMICDATABASE. COMPUT. BIOMED. RES. 27, 97–115.

STEVENS, R.D., ROBINSON, A.J. & GOBLE, C.A. (2003). MYGRID: PERSONALISED BIOINFORMATICS ON THE INFORMATION GRID. BIOINFORMATICS, 19:I302–I304.

SCHONBACH, C., KOWALSKI-SAUNDERS, P. & BRUSIC, V. (2000). DATA WAREHOUSING IN MOLECULAR BIOLOGY. BRIEFINGS IN BIOINFORMATICS, 1(2), 190-198(9).

SHETH, A. & LARSON, J.A. (1990). FEDERATED DATABASE SYSTEMS FOR MANAGING DISTRIBUTED, HETEROGENEOUS, AND AUTONOMOUS DATABASES. ACM COMPUTING SURVEYS 22 (3), 183-236.

STEIN, L. (2003). INTEGRATING BIOLOGICAL DATABASES. NAT. REV. GEN. 4, 337-345.

STEVENS, R., GOBLE, C.A., BAKER, P. & BRASS, A. (2001). A CLASSIFICATION OF TASKS IN BIOINFORMATICS. BIOINFORMATICS 17, 180–188.

ARUMUGAM, S. & CHAKRAPANI, L.N. RECONCILIATION AND MERGING OF HETEROGENEOUS XML DATA.

MALIK, T., SZALAY, A.S., BUDAVARI, T. & THAKAR, A.R. (2002). SKYQUERY: A WEB SERVICE APPROACH TO FEDERATE DATABASES. ARXIV PREPRINT CS.DB/0211023

WANG, L., RIETHOVEN, J.-J. & ROBINSON, A. (2002). XEMBL: DISTRIBUTING EMBL DATA IN XML FORMAT. BIOINFORMATICS, 18, 1147–1148.

WHEELER, D.L., CHAPPEY, C., LASH, A.E., LEIPE, D.D., MADDEN, T.L., SCHULER, G.D. TATUSOVA, T.A. & RAPP, B.A. (2000). DATABASE RESOURCES OF THE NATIONAL CENTER FOR BIOTECHNOLOGY INFORMATION. NUCLEIC ACIDS RES., 28, 10–14.

HASSELBRING, W. (2000). INFORMATION SYSTEM INTEGRATION, COMMUNICATIONS OF THE ACM 43, 32-38.

LITWIN, W., MARK, L. & ROUSSOPOULOS, N. (1990). INTEROPERABILITY OF MULTIPLE AUTONOMOUS DATABASES. ACM COMPUTING SURVEYS (CSUR), 22(3), 267–293.

ZDOBNOV, E. M., LOPEZ, R., APWEILER, R. & ETZOLD, T. (2002). THE EBISRS SERVER — NEW FEATURES. BIOINFORMATICS 18, 1149–1150.

CHAMPION, M.F., NEWCOMER, C. & ORCHARD, E.D. (2002). WEB SERVICES ARCHITECTURE

FLOSS, K. (2004). ORACLE SQL TUNING & CBO INTERNALS

ORACLE DATABASE PERFORMANCE TUNING GUIDE

PENNINGTON, C., CARDOSO, J., MILLER, J.A., PATTERSON, R.S., & VASQUEZ, I. (2007). ″WEB SERVICES AND SERVICE ORIENTED ARCHITECTURES″, IN "SEMANTIC WEB SERVICES: THEORY, TOOLS AND APPLICATIONS″, IDEA GROUP.

HAAS, L.M., LIN, E.T. & ROTH, M. (2002). A.:DATA INTEGRATION THROUGH DATABASE FEDERATION. IN: [IBM02], 578-596

MARKOWITZ, V.M., CHEN, I.A., KOSKY, A.S. & SZETO, E. (1999). OPM: OBJECT-PROTOCOL MODEL DATA MANAGEMENT TOOLS'97. IN: LETOVSKY SI, EDITOR. BIOINFORMATICS DATABASES AND SYSTEMS. DORDRECHT: KLUWER ACADEMIC PUBLISHERS

SIVASHANMUGAM, K., VERMA, K., SHETH, A. & MILLER, A.J. (2003). ADDING SEMANTICS TO WEB SERVICES STANDARDS. IN: PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON WEB SERVICES

APPENDIX

A1: execution plan and statistics information of the two SQL queries:

13 rows selected.

Elapsed: 00:00:00.28

Execution Plan

0		SELECT STATEMENT (REMOTE) Optimizer=ALL_ROWS (Cost=293 Card=41 Bytes=4141)							
1	0		TABLE ACCESS (BY INDEX ROWID) OF 'TAXONNAME' (TABLE) (Cost=4 Card=1 Bytes=44)						PLASMOPR.DB.CBIL .UPENN.EDU
2	1			NESTED LOOPS (Cost=293 Card=41 Bytes=4141)					
3	2				NESTED LOOPS (Cost=149 Card=36 Bytes=2052)				
4	3					NESTED LOOPS (Cost=113 Card=36 Bytes=1692)			
5	4						TABLE ACCESS (FULL) OF 'QUERY_RESULT_2' (TABLE) (Cost=5 Card=18 Bytes=252)		PLASMOPR.DB.CBIL .UPENN.EDU
6	4						TABLE ACCESS (BY INDEX ROWID) OF 'NAFEATUREIMP' (TABLE) (Cost=6 Card=2 Bytes=66)		PLASMOPR.DB.CBIL .UPENN.EDU
7	6							INDEX (RANGE SCAN) OF 'NAFEATUREIMP_IND03' (INDEX) (Cost=2 Card=29)	PLASMOPR.DB.CBIL .UPENN.EDU
8	3					TABLE ACCESS (BY INDEX ROWID) OF 'NASEQUENCEIMP' (TABLE) (Cost=1 Card=1 Bytes=10)			PLASMOPR.DB.CBIL .UPENN.EDU
9	8						INDEX (UNIQUE SCAN) OF 'PK_NASEQUENCEIMP' (INDEX (UNIQUE)) (Cost=0 Card=1)		PLASMOPR.DB.CBIL .UPENN.EDU
10	2				INDEX (RANGE SCAN) OF 'TAXONNAME_IND01' (INDEX) (Cost=2 Card=1)				PLASMOPR.DB.CBIL .UPENN.EDU

Statistics

---

0 recursive calls

0 db block gets

0 consistent gets

```
      0  physical reads
      0  redo size
   1360  bytes sent via SQL*Net to client
    512  bytes received via SQL*Net from client
      2  SQL*Net roundtrips to/from client
      0  sorts (memory)
      0  sorts (disk)
     13  rows processed
```

The execution plan and statistics information of second SQL query:

13 rows selected.

Elapsed: 00:00:06.75

Execution Plan

0		SELECT STATEMENT Optimizer=ALL_ROWS (Cost=1013 Card=529 Bytes=87814)							
1	0		SORT (ORDER BY) (Cost=1013 Card=529 Bytes=87814)						
2	1			HASH JOIN (Cost=1012 Card=529 Bytes=87814)					
3	2				NESTED LOOPS (Cost=582 Card=524 Bytes=41920)				
4	3					NESTED LOOPS (Cost=57 Card=524 Bytes=28296)			
5	4						TABLE ACCESS (FULL) OF 'QUERY_RESULT_2' (TABLE) (Cost=2 Card=18 Bytes=252)		
6	4						REMOTE* (Cost=3 Card=29 Bytes=1160)		PLASMOPR.DB.CBIL .UPENN.EDU
7	3					REMOTE* (Cost=1 Card=1 Bytes=26)			PLASMOPR.DB.CBIL .UPENN.EDU
8	2				REMOTE* (***Cost=429 Card=25987 Bytes=2234882***)				PLASMOPR.DB.CBIL .UPENN.EDU
6	SERIAL_FROM_R						SELECT		

	EMOTE		"NA_SEQUENCE_ID","SOURCE_ID" FROM "DOTS"."GENEFEATURE" "GF" WHERE "SOURCE	
7	SERIAL_FROM_REMOTE		SELECT "NA_SEQUENCE_ID","TAXON_ID" FROM "DOTS"."NASEQUENCE" "S" WHERE :1="NA_SEQ	
8	SERIAL_FROM_REMOTE		SELECT "TAXON_ID","NAME","NAME_CLASS" FROM "SRES"."TAXONNAME" "TN" WHERE "NAME_C	

Statistics

---

14 recursive calls

0 db block gets

5 consistent gets

0 physical reads

0 redo size

1360 bytes sent via SQL*Net to client

512 bytes received via SQL*Net from client

2 SQL*Net roundtrips to/from client

1 sorts (memory)

0 sorts (disk)

13 rows processed

A2: execution plan and statistics information of the SQL query with “hint”:

13 rows selected.

Elapsed: 00:00:00.35

Execution Plan

0		SELECT STATEMENT (REMOTE) Optimizer=ALL_ROWS (Cost=290 Card=41 Bytes=4469)							
1	0		TABLE ACCESS (BY INDEX ROWID) OF 'TAXONNAME' (TABLE) (Cost=4 Card=1 Bytes=44)						PLASMOPR.DB.CBIL .UPENN.EDU
2	1			NESTED LOOPS (Cost=290 Card=41 Bytes=4469)					
3	2				NESTED LOOPS (Cost=146 Card=36 Bytes=2340)				
4	3					NESTED LOOPS (Cost=110 Card=36 Bytes=1980)			
5	4						REMOTE* OF 'QUERY_RESULT_2' (REMOTE) (Cost=2 Card= !18 Bytes=396)		
6	4						TABLE ACCESS (BY INDEX ROWID) OF 'NAFEATUREIMP' (TABLE) (Cost=6 Card=2 Bytes=66)		PLASMOPR.DB.CBIL .UPENN.EDU
7	6							INDEX (RANGE SCAN) OF	PLASMOPR.DB.C

								'NAFEATUREIMP_IND03' (INDEX) (Cost=2 Card=29)	BIL .UPENN.EDU
8	3					TABLE ACCESS (BY INDEX ROWID) OF 'NASEQUENCEIMP' (TABLE) (Cost=1 Card=1 Bytes=10)			PLASMOPR.DB.C BIL .UPENN.EDU
9	8						INDEX (UNIQUE SCAN) OF 'PK_NASEQUENCEIMP' (INDEX (UNIQUE)) (Cost=0 Card=1)		PLASMOPR.DB.C BIL .UPENN.EDU
10	2				INDEX (RANGE SCAN) OF 'TAXONNAME_IND01' (INDEX) (Cost=2 Card=1)				PLASMOPR.DB.C BIL .UPENN.EDU
5	SERIAL_FROM_REMOTE							SELECT "SOURCE_ID","RESULT_INDEX_COLUMN" FROM "SAMWZM"."QUERY_RESULT_2" "A4"	

Statistics

---

7 recursive calls

0 db block gets

4 consistent gets

0 physical reads

0 redo size

```
1360  bytes sent via SQL*Net to client
 512  bytes received via SQL*Net from client
   2  SQL*Net roundtrips to/from client
   0  sorts (memory)
   0  sorts (disk)
  13  rows processed
```

CHAPTER 3

SEMI-AUTOMATIC COMPOSITION OF WEB SERVICES FOR THE BIOINFORMATICS DOMAIN[9]

9. Zhiming Wang, Rui Wang, Cristina Aurrecoechea, Douglas Brewer, John A. Miller, Jessica C. Kissinger, Eileen T. Kraemer. To be submitted to International Journal of Web and Grid Services (IJWGS). Address: Department of Computer Science, University of Georgia, Athens, GA 30602. E-Mail:jam@cs.uga.edu. Telephone: 706-542-3440. Fax: 706-542-2966.

1. ABSTRACT

Web service technology is becoming an increasingly popular way to provide data and computational resources in the biological domain, in part because of the advanced features it provides such as language independence, platform independence, ability to discover services and ease of Web-based programmatic access. As the number of Web services provided by the bioinformatics community continues to grow, the need to use these services in combination is becoming more apparent. Typically, Web services are composed to form a process or workflow using a graphical design tool, such as a BPEL designer. However, for the scientific community, use of these BPEL tools requires a high-level of expertise. In the biological domain, there are some research efforts geared towards providing easier ways to compose biological Web services, most notably the interrelated BioMoby/Taverna projects. So far, the tools developed by such projects are still too difficult to be widely used by the average biologist. Furthermore, they tend to not fully follow Web services standards. Therefore, providing tools that lower the learning curve for Web service composition, while still complying with Web service standards, could greatly benefit biologists and bioinformaticians. In this paper, we describe WS-BioZard, a comprehensive framework for addressing this need by using wizards and semantics to provide a suite of tools that support practical semi-automatic composition of Web services in the biological domain.

2. INTRODUCTION

Over the last two decades, substantial progress has been made in the development of scientific workflows. One of the keys to make this technology useful is to have a universal infrastructure with seamless interoperability and integration capabilities. In the 1990's, the Common Object Request Broker Architecture (CORBA) advanced the state-of-the-art, but fell

short in providing seamless integration and it was somewhat difficult to use [31]. The new candidate for a universal infrastructure is Web service technology. Web service technology allows operations (part of a remote program) to be invoked over the Web. Besides being universal by using XML and Internet protocols, this technology has the advantage of being easier to use and more loosely coupled than CORBA, thus allowing for easier integration.

Fundamentally, a Web service is located on the Web (has a Uniform Resource Locator (URL)) and provides one or more operations to client programs. The capabilities of a Web service are described by the Web Service Description Language (WSDL) [32] so that potential clients know how to invoke the operations and to facilitate the discovery of Web services. A service is invoked simply by sending it an XML request message and it responds with an XML response message. These XML messages are usually sealed in a Simple Object Access Protocol (SOAP) [33] envelope that puts the XML message in the body of the overall SOAP message along with additional information in a SOAP header. Note, Web services following the Representational State Transfer (REST) [34] approach eliminate the overhead of SOAP messages and send the XML messages directly.

There are several ways to discover Web services. Users may browse known sites such as XMethods (http://www.xmethods.net/ve2/index.po) or StrikeIron (http://www.strikeiron.com/) to find services. A more automatic approach is to use a Web service registry. These registries typically support the Universal Description Discovery and Integration (UDDI) standard [35], which has been used in the bioinformatics field. For example, the well-known project, myGrid (http://www.mygrid.org.uk/), a research project utilizing Grid framework to serve the life science, developed UDDI-M [49], an extension of UDDI registry for bioinformatics Web services with capability to register metadata such as location, ownership, version, etc.

Web service technology is beginning to be widely used in the scientific community [36]. In the biology domain, there are 1620 bioinformatics Web service registered in BioMoby service central (http://moby.ucalgary.ca/cgi-bin/getServiceDescription) and many service providers are listed on myGrid's website (http://www.mygrid.org.uk/wiki/Mygrid/BiologicalWebServices). After service discovery, the next step is to string together several Web services to accomplish a larger goal or task. This is done by composing Web services into a process in which the execution of the individual Web services are coordinated by a process execution engine. With Web service composition, we come full circle back to workflows. Note, at one time, workflow also included the notion of human tasks with associated work-lists, but with the addition of WS-Human Task to Web service processes, the two technologies overlap considerably. The only major difference is that Web service processes are Web scale. Processes are defined using a process definition language such as the XML Process Definition Language (XPDL) for workflows [37] and the Business Process Execution Language (BPEL) for Web service compositions. Although Web service technology is easier to use than CORBA, the process language, BPEL, is unfortunately more complex than XPDL (these two languages are briefly compared later in this paper).

Due to the complexity of defining processes to execute on the Web and the complexity of BPEL in particular, it is very difficult to directly write a correct BPEL specification. Since a BPEL specification is just an XML document one could attempt to do this, but it is not recommended. Today, BPEL specifications are created using Graphical User Interfaces (GUI), in which BPEL constructs are dragged from a toolbar, positioned on a design canvas and wired together to form a process definition. This is much simpler than manual editing. The design tools also provide some rudimentary correctness checking capabilities. Still, to create a BPEL process

to simply invoke two Web services will likely require more than ten BPEL constructs to be positioned on the canvas. Each construct will typically have several properties to be filled in. In other words, designing a BPEL process is still difficult.

This state of affairs may be appropriate for businesses that can utilize an Information Technology (IT) department, but represents an impediment for scientific communities. There are several ways to reduce the complexity of designing Web service processes. One way would be to have a simpler process definition language. Several years back, this was the case, with two of the main languages being the Web Service Flow Language (WSFL) from IBM [38] and XLANG from Microsoft [39]. These two organizations worked together (with other partners) to combine these languages to form BPEL, a universal infrastructure is of paramount importance. This combined language promotes the important goal of a universal infrastructure, but resulted in a more complex language. Still, even with a simpler language, designing process is inherently difficult.

A better course of action is to automate away some of the complexity of designing a Web service process. Automatic programming [40], in which the user tells the computer what he/she wants and the computer writes the program for the user, has been a goal of Computer Science that began shortly after the development of high-level programming languages. Some believe that such endeavors are folly, while others argue that it makes sense to chip away at the problem. In the field of Web services, there has been substantial and ongoing research on automatic composition of Web services. Since composing Web services can be viewed as programming in the large (the detailed code is already written in the implementations of the services), the problem is reduced to connecting the services together in a sensible manner (analogously to how a robot would move and stack blocks to achieve an overall goal). Thus, if one were able to

provide unambiguous semantics for each of the operations within a Web service that includes the types of inputs and outputs as well as their preconditions and effects, one could try to use AI planning technology to solve the problem [1, 2, 3]. In simple cases, this may work. However, once the data types and control flow become complicated (e.g., with conditional branching, loops and parallel execution), the ability to create a correct and useful process degrades [4, 5]. At the present time, it seems that we need automation working alongside with human intelligence. This is the research area of semi-automatic Web service composition. This paper reports on progress made in developing a prototype tool, called WS-BioZard, that supports the semi-automatic composition of Web services[10].

The target area for our work is scientific workflows. In particular, WS-BioZard was created to support the on-the-fly composition of Web services provided by the ApiDB/EuPathDB project (http://eupathdb.org/) [41] as well as related services at other sites, such as the National Center for Biotechnology Information (NCBI), the European Bioinformatics Institute (EMBL-EBI), the DNA Data Bank of Japan (DDBJ) and the Protein Data Bank of Japan (PDBJ). The ApiDB project tries to provide biologists with uniform, integrated and centralized Web access to eukaryotic pathogen genomes from *Cryptosporidium*, *Giardia*, *Plasmodium*, *Toxoplasma* and *Trichomonas*. Our target audience is therefore biologists and bioinformaticians. With the advanced features provided by Web services, such as language independence, platform independence and ease of programmatic access, Web services are increasingly being used in bioinformatics to provide access to data sources as well as software to analyze the data. While individual bioinformatics Web services are useful, situations often arise where more than one service is required to perform a complete biological analysis. For example, whenever a new

10. In general, when creating complex processes that involve multiple Web services, users may choose between (1) manually composing the services themselves using a GUI-based design tool, (2) using fully automatic composition software, or (3) using the hybrid approach of semi-automatic composition.

Genomic nucleotide sequence is annotated, biologists try to understand its functionality through searching for homologous sequences. Homologous sequences are related by evolutionary descent and are often, but not necessarily, similar at the primary amino acid sequence level. This task can be achieved through composing biological Web services: First a BLASTX[11] Web service could be used to identify amino acid sequences from a well-known protein sequence database. Then another Web service is used to retrieve the 'good hits' amino acid sequences. A third bioinformatics Web service is used to identify protein domains that will help to determine the functionality.

Without the help of a Web service composition tool, such tasks would be time consuming and error-prone (typically a biologist would use a copy/paste approach, as described in section 7) or would require good programming skills. Compared with a traditional programming-based approach, the Web service composition approach has the following benefits:

- It is easier to develop, given a good-quality design tool.
- It is easier to change a Web service during the composition, through drag and drop in a design tool.
- It is easier to reuse, since the resultant process can itself be invoked as a standard Web service.

However, for biologists, creating biological Web service compositions using current design tools is full of challenges. In this paper, we describe WS-BioZard, a new and comprehensive framework for lowering the complexity of Web service composition using a semi-automatic

11. The Basic Local Alignment Search Tool (BLAST) can be used to infer gene family information and sequences' relationship, such as function or evolution, through comparing the querying sequence to sequence databases and calculating the statistical significance of matches. BLASTX is a special BLAST program to search protein databases using a translated nucleotide query.

approach. WS-BioZard utilizes Semantic Annotations for WSDL (SAWSDL:*www.w3.org/2002/ws/sawsdl/*) and domain ontologies to add semantics to Web services. This allows us to design and implement (1) a service discovery tool with a sophisticated User Interface (UI), and (2) a service composition tool with an intuitive graphical UI and a multi-faceted UI Wizard having data mediation capabilities. The result of the composition is a BPEL process. To simplify usage further, WS-BioZard can automatically deploy the BPEL process as a standard Web service in a BPEL-compliant engine. A GUI client tool for invoking the BPEL process is also provided with WS-BioZard to invoke the BPEL process.

The remainder of this paper is structured as follows: In section 3, a brief review of existing XML-based composition languages is given. Due to the complexity of composition languages, we discuss the use of semi-automatic composition techniques to simplify the composition process from the perspective of control flow and data flow by using tools to assist the human designer. Section 4 covers related work in semi-automatic composition as well as composition techniques used in bioinformatics. Data mediation plays an important role in simplifying the creation of BPEL processes and is presented in section 5. Section 6 explains the design and implementation of WS-BioZard, which follows a layered approach. Although WS-BioZard is principally a design tool, the related issues of deployment and execution are also discussed in this section. Section 7 compares WS-BioZard with several other Web service composition tools. Finally, section 8 gives our conclusions and future work.

3. SEMI-AUTOMATIC COMPOSITION

The main goal of WS-BioZard is to reduce the complexity of designing Web service processes. It is intended to be a designer only and not a process/workflow engine. Therefore, it

produces design artifacts representing the composition that are fed to an execution engine. Today, these artifacts are almost always XML documents. Currently, there are five popular process/workflow definition languages that are open standards: XML Process Definition Language (XPDL) [37], Web Services Business Process Execution Language (WS-BPEL) [42], Web Service Choreography Definition Language (WS-CDL) [43], Business Process Modeling Language (BPML) [44] and Unified Modeling Language (UML) [45]. Although Web service composition has benefited from experiences in the workflow domain, the standards in the two domains are distinct. The five languages fall into two categories: those that are oriented toward Web services, such as WS-BPEL and WS-CDL, and those that are more generally for business processes/workflows, including XPDL, BPML and UML.

In comparison to a workflow, Web service composition is based on interactions in the context of the Service-Oriented Architecture (SOA) [46]. Web service composition specifies an XML grammar that can be seen as a programming language to combine control and data flow with invocation of the services.

Web service composition falls into two main categories: orchestration and choreography. Orchestration corresponds to the traditional view of defining processes by specifying the control and data flow. Choreography takes a more collaborative view and focuses on defining the message exchange [6, 7]. Although several languages have been developed, the industry has coalesced on two open standards: WS-BPEL and WS-CDL.

- As an orchestration language, OASIS standardized the Web Service Business Process Execution Language (WS-BPEL 2.0) in 2007 (based on the BPEL4WS submission co-written by IBM, Microsoft, BEA, SUN and Oracle in 2003). It specifies how Web

services can be composed using process flow constructs for sequential, conditional, iterative and parallel execution, etc.

- As a choreography language, W3C standardized the Web Service Choreography Definition Language (WS-CDL) in 2004. It specifies how Web services can collaborate with each other. To join a collaboration, a Web service must follow the protocol (types and orderings of messages) specified in a WS-CDL documents.

Compared with WS-CDL, WS-BPEL currently has more support from industry and wider acceptance from users. Also, there are more WS-BPEL engines, (e.g., Active Endpoints' ActiveBPEL Engine, Apache ODE (Orchestration Director Engine), Sun's BPEL Service Engine, Oracle BPEL Process Manager and IBM's WebSphere Process Server) than WS-CDL engine, and more graphical design tools for BPEL (e.g., Active Endpoints' ActiveBPEL Designer, Eclipse BPEL Designer, Sun's NetBeans BPEL Designer, Oracle's JDeveloper BPEL Designer and IBM's WebSphere Integration Developer) than for WS-CDL.

Other related workflow languages have been developed such as YAWL [8] and XScufl [9]. XScufl is an ongoing collaborative project between EBI, IT Innovation and the Human Genome Mapping Project (HGMP) of the Medical Resource Center (MRC) and is used by Taverna, a bioinformatics tool [25]. In comparison with the above main Web service composition languages, these two have smaller tool support bases.

The choice of an SOA-oriented process definition language for WS-BioZard boils down to a trade-off between tool support and complexity. Although it is more complex, WS-BPEL's tool support is vastly greater. On the positive side, WS-BPEL is feature rich. It implements thirteen of the original twenty workflow patterns [10], including the five most basic patterns: sequence,

parallel split, synchronization, exclusive choice, and simple merge. To implement these workflow patterns, it provides many rich control flow constructs commonly associated with programming languages. Thus, we are still left with our original problem of reducing complexity, made even more critical due the complexity of BPEL. Even with a good GUI designer, creating a correct BPEL specification is very difficult for non-IT professionals. Besides the complexity of control flow and all the workflow patterns supported, data flow can also be complex. In many fields and in biology in particular, a high level of heterogeneity is found in the types of data that flow between the diverse services in a BPEL process. According to [11], there are twenty different representations for DNA sequence alone. BPEL allows data to be transformed in assign/copy statements using the XPath language [47]. Yet even for the IT professional, writing XPath expressions to transform the output of one service into the input of the next service is difficult and error prone (this is an actual weakness of the BPEL standard).

With manual composition using a GUI-based BPEL designer being too difficult for our target user, one could attempt to develop a tool that fully supports automatic composition. Fully automatic planners exist and their use in Web service composition has been studied in [12, 13]. The commonality between all the planning methods is that they require preconditions and effects to be specified for their correct operation. In the context of planning, a precondition specifies what should be true for correct invocation of the service, and an effect specifies how the invocation of the service changes the state of the process (via its outputs and side-effects). The control flow is determined by the planner by matching the preconditions and effects to find services that can provide necessary transformations leading to a goal state. The data mediation may be provided by a planner trying to find a data transformation, which may be provided or calculated with or without

annotations. The good thing about automatic composition is that since the user does not have to deal with control or data flow, the complexity of WS-BPEL can be hidden from the user.

However, there exist some downfalls to automatic composition. The problem with automatic composition is two-fold: it relies on the availability of complete formal representations of (1) the domain knowledge, and (2) the individual cases that need to be resolved, i.e., their initial state and goal state need to be formally encoded. The task of formally specifying a domain in sufficient fidelity so that it can be used for automated planning presents a huge challenge, especially in the biology domain that is advancing so rapidly that it is a moving target. Incomplete domain knowledge will often result in a situation in which an automated planner fails to produce a plan. As for specification of inputs, outputs, preconditions and effects of the Web services used in the composition, these planners rely on their semantic annotation. Unfortunately, specifying complex semantics (e.g., effects) highly accurately is in itself a challenge. Moreover, research has shown they could be good at composing sequences of services, but when we introduce the need for loops and particularly the last two basic workflow patterns (Arbitrary Cycles and Implicit Termination), they have not been shown to work well [14].

Having seen the shortfalls in the manual and fully automatic approaches, WS-BioZard implements a semi-automatic approach. While still relying on semantically annotated Web services, it overcomes the weaknesses of automatic composition because it allows users to draw upon their experience within the domain and compensate for missing or erroneous ontologies. WS-BioZard still provides automation when possible, helping the user along with the wizards, matching outputs with inputs and by reducing the number of available patterns to those deemed essential. In particular, our approach provides nine of the original twenty patterns, including the five most basic patterns. The patterns we lose, when compared to WS-BPEL, are deferred

choice, interleaved parallel routing, cancel task, and cancel case[12].

We close this section by introducing the high-level architecture of WS-BioZard as well as some of the principles upon which it is based.

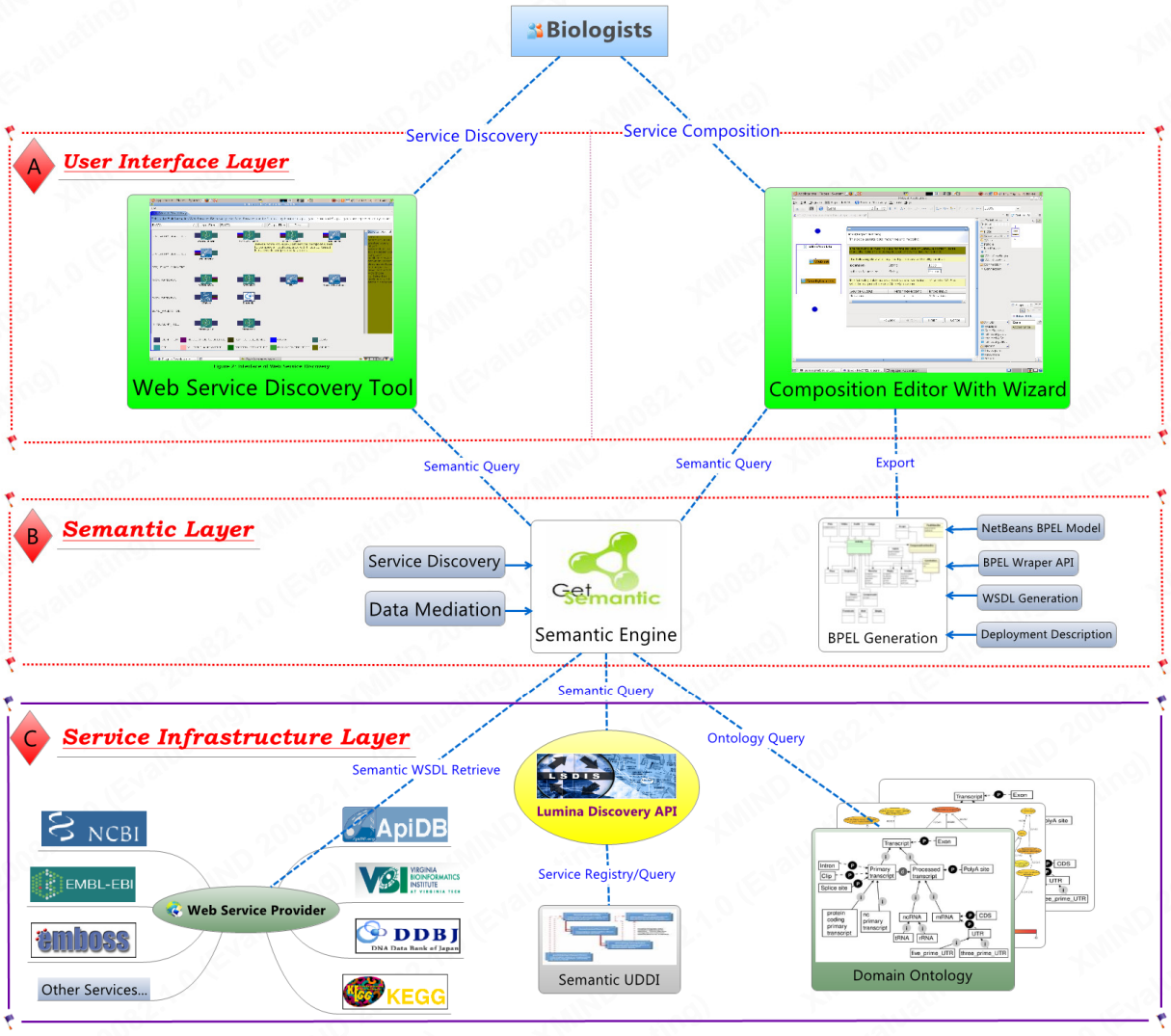

Figure 3.1. Architecture of WS-BioZard.

WS-BioZard is architected following the layered approach shown in Figure 3.1 and consists of three main layers: a user interface layer, a semantic layer, a service infrastructure layer. The

12. In our opinion, the patterns lost in our approach are of less use to our users, while we still keep much of the power of WS-BPEL with a more intuitive GUI. Thanks to the flexible architecture, we can implement those patterns when they are needed with minimum effort.

top layer consists of two GUI tools: a Web service discovery tool and Web service composition editor. The discovery tool is used to discover and browse the available Web services based on criteria specified by the end user, such as input data type or output data type. The composition editor is used to compose Web services in a semi-automatic approach with the aid of a multi-faceted wizard. These GUI tools were developed using several basic Eclipse technologies: Standard Widget Toolkit (SWT), JFace and Graphical Editing Framework (GEF), as well as newer and higher-level Eclipse technologies: Eclipse Modeling Framework (EMF) and Eclipse Graphical Editing Framework (GEF). The manner in which these technologies were used and the advantages they gave in the development of WS-BioZard are briefly discussed in the Appendix. The discovery and composition tools are described in detail in sections 6.1 and 6.2, respectively. Although WS-BioZard is built using the Eclipse platform, WS-BioZard can run as a stand-alone Java application outside the Eclipse workbench. The bottom layer is essentially existing technology from the METEOR-S project and includes both the Radiant (semantic annotation and publication of Web services) and Lumina (semantic discovery of Web services) software and tools. The middle layer serves as a bridge between the top layer GUI tools and bottom layer METEOR-S Semantic Web Services (SWS) technology. Section 6.3 discusses the Model Layer (middle layer), while section 6.4 discusses the Semantic Process Layer (bottom layer).

Currently, Web service technology lacks full support for service discovery, service invocation/interoperation, service negotiation and service composition. Adding semantics to Web services is the most promising solution for this problem. The objective of Semantic Web Services is to provide a knowledge representation of individual services in order to allow automatic machine processing. Adding Semantics to Web services can provide the following benefits:

- Better Discovery--Neither the Web Service Description Language (WSDL) nor the Universal Discovery and Description (UDDI) provides enough help to the client about what a Web service offers. A Semantic Web service describes its properties and capabilities, which can help external applications to automatically determine its purpose and pave the path for automatic composition.
- Better Interoperability/Invocation--Semantics can help mapping the data exchanged between the services to make sure invocations operate seamlessly.
- Better Composition--Automatically check data mapping and enable data mediation.

The Semantic Annotations for WSDL and XML Schema (SAWSDL) W3C Recommendation is the first step by W3C to add semantic support to Web services technology by adding semantic annotations to WSDL components. It is now poised to unleash the power, flexibility, and logic of the next generation of Web services. Within section 6.4, we discuss how we utilize SAWSDL and how it facilitates both Web service discovery and composition for WS-BioZard.

4. RELATED WORK

In this section, we discuss some related work in the areas of semi-automatic composition in general, as well as specific Web service composition for bioinformatics. We discuss several semi-automatic Web service composition projects using criteria such as: support for filtering out inappropriate services, suggesting partial plans, checking the validity of compositions, utilizing planning strategies, modeling environment (graphical or not), use of ontologies, control flow constructs supported, executability of compositions and compliance with standards such as

BPEL.

Sirin et al. [19] present a Web service composer that uses OWL-S to provide semantics for the services. The main focus of their work is to filter out the non-compatible services at each composition step, thus helping the user to select the appropriate services. While [19] offers filtering of inappropriate web services it does not check for composition validity nor offer suggestions for partial plans. On the plus side it does provide a standard output format (OWL-S) and offers a graphical modeling environment that includes the execution of the composition. Similarly, Xu et al. [20] present Dynamic Semantic Association (DSAC) which utilizes the Dynamic Semantic (DS) and Semantic Association (SA) in service matching. This work is based on SRO (an OWL-DL ontology) and focuses on utilizing a service matchmaking algorithm to provide users candidate services to select in an interactive fashion. This work supports filtering inappropriate services and achieves sequence control flow, based on a graphical user interface. However, this work cannot suggest partial plans, check the validity of compositions, or utilize a planning strategy. The composition result is not a standard and cannot be executed. Kim [21] introduced CAT (Composition Analysis Tool), a tool for interactive workflow composition. The focus of their work is to assist the user in the creation of computational workflows. The authors' work is not directly related to service composition. However, we can conceive of a computational workflow as a service composition. The activities of the workflow compare to service operations that realize data transformations. Its strength is in evaluating "well-formedness" for the composition and suggesting fixes, but does not provide for filtering nor control constructs and their modeling environment is textual. Hakimpour [22] introduced Internet Reasoning Service (IRS), a Semantic Web Services framework. One of their implementations, IRS-III, includes a tool that supports a user-guided interactive composition

approach by recommending component Web services according to the composition context. Their approach uses Web Services Modeling Ontology (WSMO) as the language to semantically describe the functionality of the Web services. IRS-III's main strength is in that it provides an execution engine, but does not provide for help in the composition process by filtering or suggesting partial plans.

The two main projects focusing on bioinformatics Web services composition are myGrid and BioMoby. The myGrid project is part of the UK government's e-Science program [23], and aims to provide a comprehensive middleware suite specifically to support data intensive *in silico* experiments in biology. The core components are based on Web service technology, which can be adopted in a "pick and mix" way for developers and tool builders to form, execute, manage and share discovery experiments. As a subproject of myGrid, Taverna [26] is a bioinformatics Web service composition editor using the XScufl language with a friendly GUI. Compared with a general BPEL editor, it is much easier and more intuitive to use. However, it has two main drawbacks. First, the workflow generated by it can not be run outside of Taverna as a standard Web service, which limits its functionality and usability. Second, the user usually needs to manually handle data mapping using a scripting language such as the Java-based Bean Shell language. This feature provides high flexibility with the cost of making Taverna hard to use by the average biologist.

BioMoby [24] is an open source research project which implements an architecture for the discovery and distribution of biological data through Web services; data and services are decentralized, but the availability of these resources, and the instructions for interacting with them, are registered in a central location called MOBY Central. MOBY Central provides an object-driven registry query system with its own object and service ontologies. The strength of

this initiative is in the service registration, discovery and transaction process with semantic information. However, BioMOBY by itself is limited in scope, in that it does not include facilities for creating Web service processes or workflows. In order to compose BioMoby services, one can use the BioMoby plug-in to Taverna and then use the graphical workflow designer provided with Taverna (an example of the use of this designer is given in section 7).

5. DATA MEDIATION

As processes consist of control and data flow, these become two avenues along which to attack the problem of complexity. We have discussed in section 4 how WS-BioZard reduces complexity by limiting the process flow constructs to the essential ones. Furthermore, some of the BPEL elements need not be explicitly placed in the process by the designer, as WS-BioZard inserts them where needed. In section 7, we will show how this results in a much smaller and simpler design diagram. WS-BioZard reduces the complexity of designing BPEL processes by introducing mediators and wizards for automating the design: either fully automatically in the case of mediators or semi-automatically in the case of wizards. In this section, we focus on the techniques used for data mediation. A more detailed description of the components making up WS-BioZard is given in section 6.

Data mediation may be the most challenging task in Web service composition. It is even worse in the bioinformatics Web service composition domain, considering the lack of a universal naming convention, extreme diversity of terms and rich data schema. Manual data mediation could be quite frustrating since one needs to map the outputs of one service to the inputs of another using, for example, XPath within BPEL assign/copy constructs. Assistance with this could pay large dividends. In WS-BioZard, we developed an algorithm that utilizes bottom-up

SAWSDL annotations to provide a practical automatic data mediation solution. This complements the data mediation approach based on top-down SAWSDL annotations presented in [17].

Data mediation is not a new issue. It has been well researched since the inception of federated databases [28, 29]. There are three layers in schema/data heterogeneity: structural, syntactic and semantic. Compared with structural and syntactic heterogeneity, semantic heterogeneity is more difficult and complex. Semantic mappings mean translating data from one data source into another, while preserving the semantics of the data [27]. Four levels of heterogeneity are listed in [17]:

- Syntactic heterogeneity: differences in the languages used for representing elements.
- Model/Representational heterogeneity: differences in the underlying models or their representations.
- Semantic heterogeneity:
  - Same name, different meaning: for example the term ID can mean either student ID or class ID.
  - Same meaning, different name: for example, lastname and familyname have the same meaning.
- Structural heterogeneity: different structures in elements. For example: company (CEOname, CEOemail, companyName) vs. company(CEO(name, email), companyName) are two elements with different structures.

Web services communicate by exchanging XML messages with each other. The types of the messages are determined by combining the WSDL message parts with any XSD schema definitions (WSDL 2.0 eliminates message parts, so types are wholly defined using XSD). One may view these types as a tree. In effect, SAWSDL allows the nodes of these type trees to be annotated (more about this in section 6.3). Two Web services can communicate if they annotate their WSDL specifications with the same or alignable ontologies. For example, if a Web service annotates the output of its Web service with concept C (typically specified in OWL) of a given ontology and another service takes this concept as input, these services can communicate. Note, this does not require the two services to use the same types, only that their types are mappable to concept C. In this way, SAWSDL allows for, yet handles, heterogeneity.

There are two ways to use annotations to establish mappings between Web services, allowing the output of one service to be mapped to the input of the next service: the top-down matching approach and the bottom-up matching approach.

5.1. THE TOP-DOWN MATCHING APPROACH

As the name implies, this approach matches the top elements (higher level in the XSD type tree) of the output and input messages. Even though the top elements conceptually match, their XSD type trees can have differences at several levels. Hence, for matching from the top to the bottom elements, there are various types of heterogeneity to be considered. The prototypes developed in [17] and [48] both use this top-down approach for their data mediation. To handle heterogeneity top-down, they use the lifting/lowering schema mapping method. They first transform (lifting schema mapping) the output message of the first Web service to an OWL instance of concept C. Next, they transform (lowering schema mapping) this intermediate OWL

instance to the appropriate type for the input message of the second Web service. Of course, one may wish to optimize out the intermediate OWL instance by composing to the two transformations. Multiple languages can be used for specifying the transformations, for example, XSLT and XQuery have been tested in this role. They provide concise transformations, but require some level of sophistication to create. Another complication of the top-down approach is that these transformations must be applied to XML messages as the process/workflow runs. There are several ways to implement this: add them as additional steps in a SOAP Engine (e.g., Axis2) pipeline ([17] experimented with this), extend BPEL to permit XSLT transformation in <assign>/<copy> elements, or use a proxy Web service. None of these approaches is particularly easy, but in the long run one of them may become the preferred solution. The following example shows the top-down approach:

Suppose the complexType Company(CEOname, CEOemail, companyName) is the output of the first web service and the complexType company(CEO(name, email), name) is the input of the second Web service. Data mediation is used to convert the output message of the first Web service to the input message of the second Web service. The top-down approach will match and map the top elements “company” with “company”. Because the structure and attributes of “company” are different in the two messages, XSLT will be used to transform the nodes to ontological concept or properties (lifting) and then fill each node of the target message schema tree (lowering) as shown in Figure 3.2.

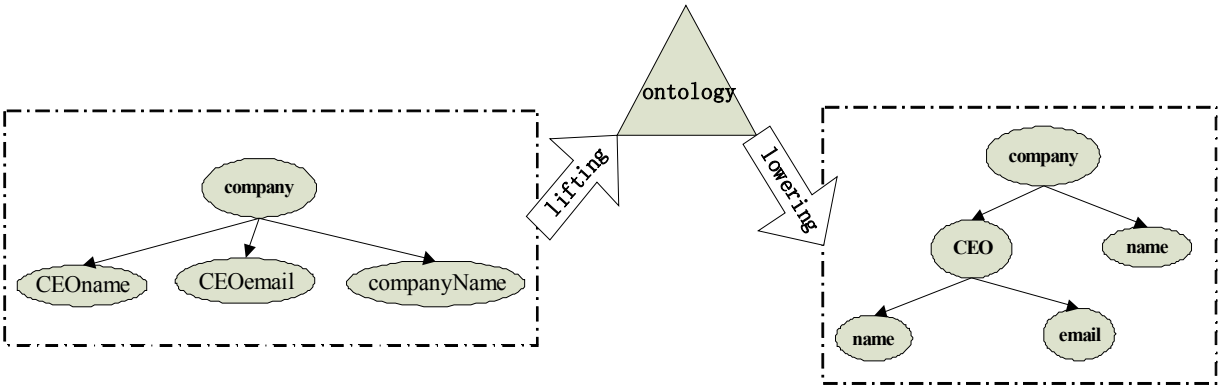

Figure 3.2. Example of Top-Down Approach.

5.2. THE BOTTOM-UP MATCHING APPROACH

The SAWSDL standard is flexible in how it allows data types to be annotated. One may specify how whole data types (e.g., an XSD complexType) can be transformed by placing the mapping at the root of the tree as in the top-down approach, or one may specify how to map the leaves of the tree as in the bottom-up approach. Assuming the whole is more than the sum of its parts, semantics may be lost in the bottom-up approach. On the other hand, it has the advantage that the mappings are simpler and in common cases should be effective. Keeping with the theme of avoiding complexity, WS-BioZard's data mediation subsystem uses the bottom-up approach.

For common cases, XPath can be used to map the bottom-level elements. These XPath expressions can then be placed in the <assign>/<copy> statements in BPEL. It is hard to imagine a simpler solution. In complex cases this may fail and we intend to test its limits as we evaluate WS-BioZard as it is put into practice. Conceptually, the lifting mapping goes into the <from> part of <copy>, while the lowering mapping goes into the <to> part (see section 6.3 for an example). These XPath expressions are automatically generated at design time from the annotated SAWSDL files for the two communicating Web services. SAWSDL model references

establish links from the XSD type tree to the ontology. The goal is simply to provide all the necessary inputs to the second Web service. Thus, for each leaf of this type tree, an XPath is generated. The model reference to the construct in the ontology is then traced back to find the conceptually matching leaf in the XSD type tree of the first services output. If the connection is made, the XPath for this part is generated. The first XPath expression is placed in the BPEL <from>, and the second XPath expression is placed in the BPEL <to>.

Algorithmically, at run time the following transformations are performed on the data being exchanged by two Web services:

$$X_i = XPath_{lower\text{-}i}(XPath_{lift\text{-}o}(X_o))$$

where X_o is the XML output message produced by the first Web service and X_i is the XML input message to be sent to the second Web service. In particular, for each leaf element in X_i, a path expression is created to lift the output and then lower it to the input. Generating these XPath expressions is, however, a bit more complex. We need to handle three cases. Suppose the first Web service's output annotates to class C_o in the ontology, and the second service's input annotates to class C_i (annotation using properties is also supported). In the first case, C_i is the same, an equivalent or a super-class of C_o, so all the input values are available from the output. In the second case, C_i and C_o have overlapping properties, so some, but not all values are available from the output. Finally, in the third case, C_i and C_o have no properties in common (unlikely to happen in practice assuming appropriate service discovery).

if $C_i = C_o$ or $C_i \equiv C_o$ or $C_i \in$ superclass(C_o) // case 1

$XPath_{lift\text{-}i}$ = path expressions generated from first services SAWSDL

$XPath_{lower\text{-}i}$ = path expressions generated from second services SAWSDL

elseif $C_i \cap C_o \neq \phi$ // case 2

$XPath_{lift\text{-}i}$ = path expressions generated from first services SAWSDL

$XPath_{lower\text{-}i}$ = path expressions created using the data mediation wizard

else // case 3

Reject WS2

endif

For case 1, WS-BioZard will automatically generate the BPEL <assign>/<copy> elements. For case 2, the data mediation wizard will popup and assist the user to produce expressions for missing values. WS-BioZard ability to automatically generate XPath expressions greatly simplifies the composition task for the user.

Let us again consider the example given in section 5.1. If we use the bottom-up approach, we will parse both of the messages' XSD types. The bottom elements for the output of the first Web service are {CEOname, CEOemail, companyName} and their annotations are {humanName, email, companyName}, respectively. The bottom elements for the input of the second web service are {name, email, name} and their annotations are {humanName, email, companyName}, respectively. Therefore, we can, as shown in Figure 3.3, match/map the elements as follows: CEOname—name, CEOemail—email, companyName—name.

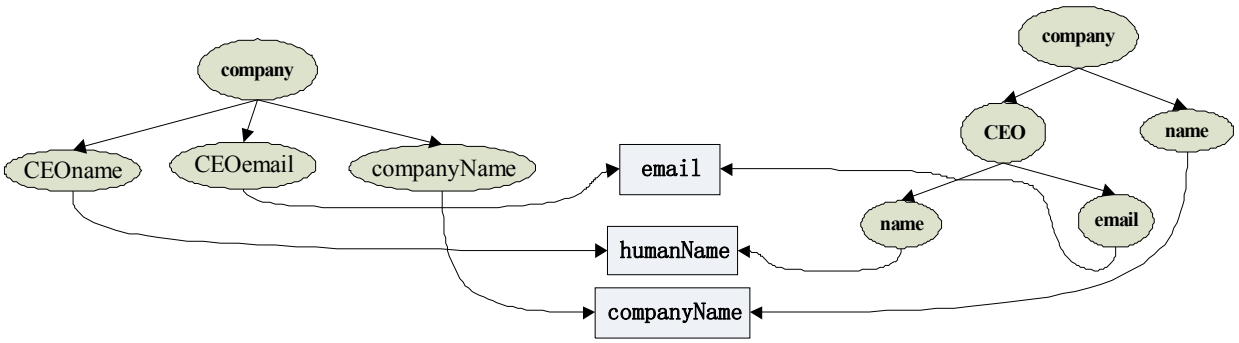

Figure 3.3. Example of Bottom-Up Approach.

The implementation details of how this data mediation works are discussed in section 6.3.

6. DESIGN AND IMPLEMENTATION OF WS-BIOZARD

In this section, we discuss the four main components of WS-BioZard: the Web Service Discovery Tool, the Web Service Composition Editor (both in the top layer), the Semantic layer (middle layer) and the Service Infrastructure layer (bottom layer). We close by discussing deployment and execution of BPEL processes.

6.1. WEB SERVICE DISCOVERY TOOL

Through the intuitive Web service discovery interface shown in Figure 3.4, biologists can view basic Web service information, such as service name, provider represented by the central icon, description, input/output data type and functionality. Furthermore, they can query/filter Web services by input or output data type. With the aid of this tool, biologists can get the overall information about all the available Web services, which will be helpful in the next step—Web service composition. There are five main regions in the tool: The main central region contains all the dynamically discovered services from a semantic UDDI located locally. The left column

consists of top-level ontological service functionalities, while the right column gives the detailed description of the selected Web service. The bottom region provides a color coded key for the services' input/output data types, while the top region is used for filtering of Web services based on, for example, input/output data types.

As shown in Figure 3.4, each service icon is composed of the service provider's logo (in the middle of the icon), the service name (at the bottom of the icon) and input/output data types represented by color coded tab and attached to the middle icon. Each color represents a type corresponding to the bottom area, and the same color shared by two services' output and input means that the two services could be composed together (i.e., there is at least a partial match). In Figure 3.4, there are ten input/output data types/file formats[13]: identifier, nucleotide_sequence, peptide_sequence, fasta, blast_output, gff, multiple_alignment, protein_structure, phylogenetic_tree and others. The input/output data type information is stored in a domain ontology and referenced by SAWSDL annotations in their XML schema part. The ontology as well as the chosen ontological functionalities and types are customizable by the user. Whenever the mouse hovers on a service, the description of this service will popup. When a service is selected, the detailed description of this service is shown in the right column. At the same time, all the incompatible services[14] are automatically filtered out. Furthermore, biologists can filter out Web service by input/output data type using features given in the top region.

The services are dynamically retrieved based on queries to a semantic UDDI registry, which contains all the registered WSDLs with SAWSDL annotations. The services are arranged by rows, with each row representing a functional category (or top-level functional concept) from the given

13 A file format could also be used as the input or output of a Web service.

14. The input of these services is not compatible/matched with the output of the selected one. In other words, the incompatible services can not be composed with the selected service because the input and output can not match.

ontology. Currently, there are seven categories defined in the ontology used in WS-BioZard: “similarity_searches”, “sequence_convert”, “data_retrieval”, “phylogenetic_tree”, “gene_prediction”, “multiple_alignment”, “structural_analysis”. Whenever there is a need to extend with a new functional category, the only place to be modified is the ontology. Once the new functionality category is properly inserted into the ontology, it can be automatically shown and used by the service discovery tool[15]. Any service annotated with this new functionality will be shown in the Web service discovery interface automatically, as long as it is registered with a semantic UDDI[16].

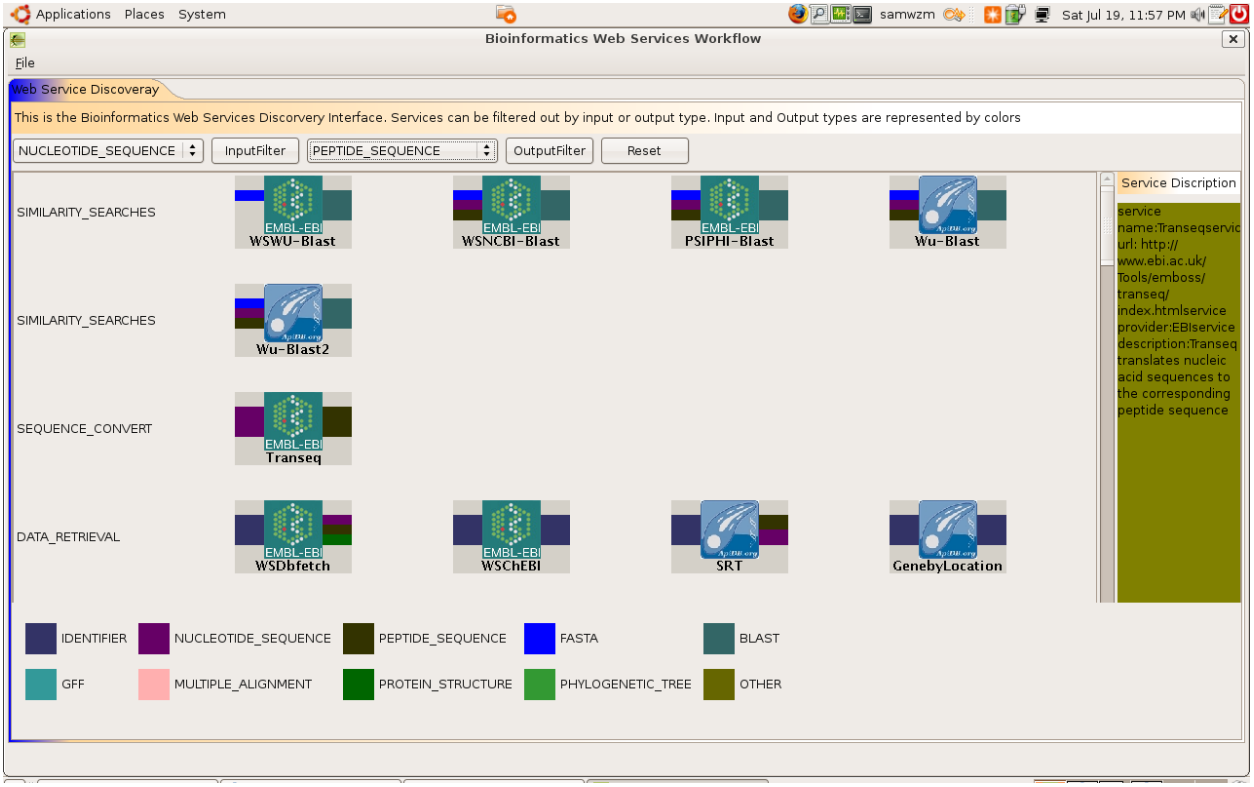

Figure 3.4. Web Service Discovery Tool.

15. In the future, the service discovery tool will be more configurable. The advanced user can choose ontology and select functional categories and data types to customize the service discovery tool.

16 In the future, the Web services in the discovery tool could be dragged and dropped into the service composition tool (Section 6.2), and the input/output information would be more utilized in the service composition tool.

6.2. WEB SERVICE COMPOSITION EDITOR

In this subsection, we discuss how the Composition Editor utilizes a multi-faceted wizard to assist biologists in composing Web services. We then discuss how our editor simplifies the service composition process.

6.2.1. A MULTI-FACETED WIZARD

A wizard is a user interface element designed to assist end users with a specific task. A well-defined wizard should hide much of the complexity involved in the task from the user. At the same time, it should provide a supplementary rather than substitutive way to accomplish the task to avoid restricting functionality for advanced users [15]. The end user is led through a sequence of dialogs to collect all the necessary information and user input for the task. A general purpose wizard may be too complex and difficult to implement. However, a wizard for a specific domain can be very useful and can lower a user's burden, especially when performing a complex task [16]. Restricting the wizard to a specific domain within bioinformatics allows the wizard to utilize one or a few related domain ontologies. Even though complex and risky alignments of diverse ontologies can be avoided, the difficulty still exists due to the large number of complex ontologies in the biological domain.

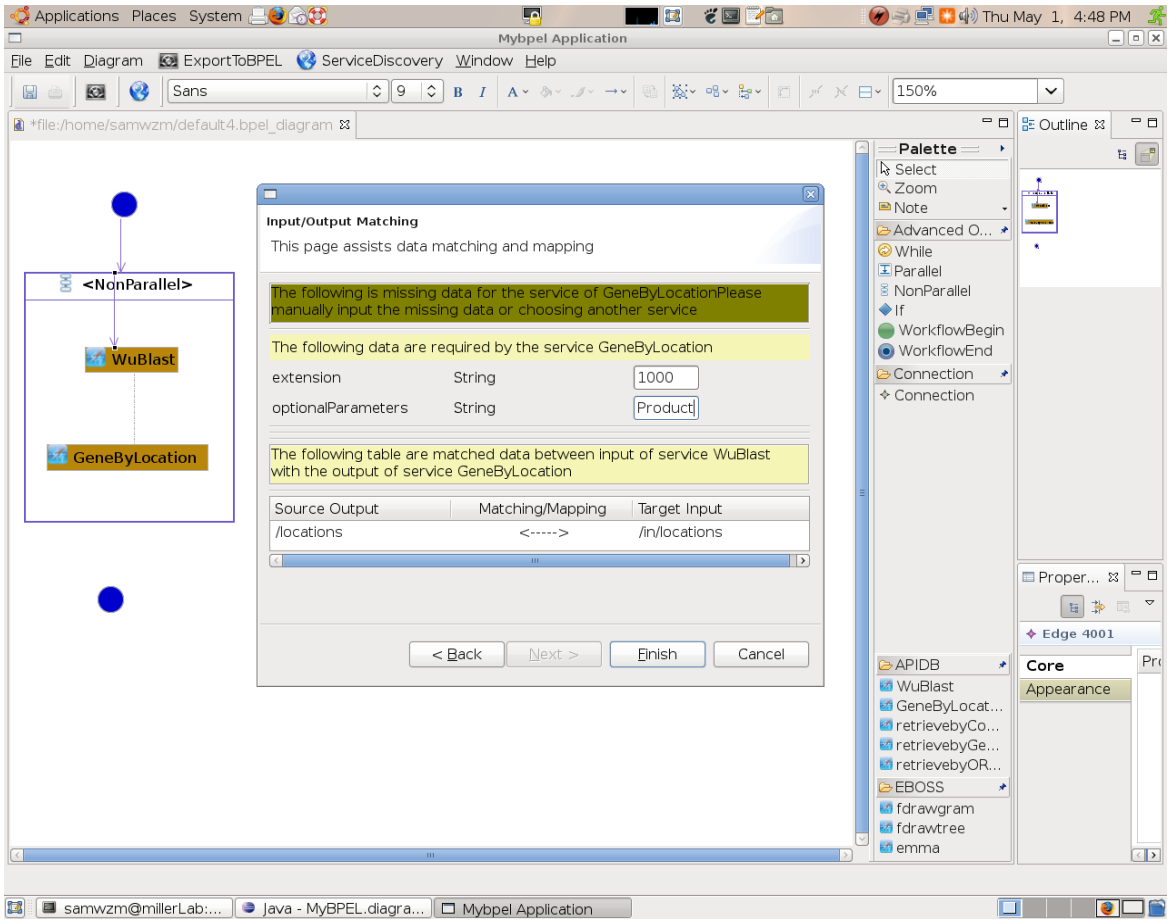

Figure 3.5. Our BPEL Editor with Wizard.

In our framework, the role of the wizard is to be a biologist's assistant. For example, a wizard can automatically pop up when a biologist tries to connect any two services, providing information about service input, output and functionality. Furthermore, a wizard can be used to check the services mapping/matching as well as assist in data mediation. Because of the above benefits provided by our wizard, we believe that our architecture significantly reduces the complexity of bioinformatics Web services composition. Figure 3.5 shows the wizard assisting in data mapping/matching when connecting two web services (WuBlast and GenesByLocation in our example), in our Web service composition editor. There are two parts in the wizard, the top part is used to handle the missing values (the message in the input of Web service two cannot

match with any message from the output of Web service one). After the user types in the missing values, WS-BioZard can automatically generate the corresponding “assign”, “copy”, “from”, “to” and “variable” without a line of code from the end user. In the next release of WS-BioZard, we plan to provide a sample value, the data type of the missing value and a semantic annotation to further assist the user. The bottom part shows the matching information between output messages in the first Web service and input messages in the second Web service.

6.2.2. SIMPLIFYING SERVICE COMPOSITION

Since BPEL is a very general orchestration language, most BPEL editors will reflect this through an interface with all 13 BPEL constructs/workflow patterns. This makes the editor too complex to be willingly used by the average biologist. In our composition editor, multiple approaches are used to lower the learning curve for biologists.

First, we make the palette more intuitive by separating the Web service selection from the flow control activities and hide most of the BPEL elements from the end user. For example, there are no “receive”, “reply”, “partnerLink”, “assign” or “invoke” elements in the BPEL editor canvas. There are two reasons why we choose this approach:

- End users can focus more on business logic, instead of detailed (and often subtle) issues of process flow.
- Those elements have a clear meaning for computer science users. However, they may not be easy to understand by biologists. By hiding these elements from the users and letting WS-BioZard automatically handle these concepts, the learning curve for WS-BioZard is significantly reduced.

This simplification for end users (biologists) comes with the cost of implementation complexity for us (the developers). For general BPEL editors such as ActiveBPEL, NetBeans BPEL and Eclipse BPEL, each BPEL element shown in the editor canvas corresponds to its implementation in the BPEL model. Whenever an end user modifies a BPEL process (either insert or delete), it needs less work to keep synchronization between the BPEL process shown in the canvas with the BPEL model. However, in our approach, we must implement a bridge between the BPEL process shown with the underlying full BPEL model. This bridge is used to keep synchronization between our BPEL editor with the BPEL model. For example, with WS-BioZard, when a user drags-and-drops a service into the editor, a namespace, input/output variables, partnerlink, assign and invoke will be automatically generated (see section 6.3.3 for details). Second, whenever there is a connection between two services, a wizard will popup and collect information from a user to create BPEL activities, such as “assign”, “copy” and “invoke”. Whenever the connection is reoriented or deleted, the corresponding actions will be taken to make the correct change. All these activities are transparent to the end user. Third, each possible Web service selection actually represents a service operation, for the following three reasons:

- Eventually an operation will be chosen to invoke the service.
- Many Web services include multiple operations and the end user is often more interested in a specific operation rather than the overall Web service itself.
- Automatic data matching/mapping can only be achieved after the operation is finalized.

6.3 THE SEMANTIC LAYER

The top layer consists primarily of Graphical User Interface code, while the bottom layer primarily supports the registration and discovery of semantic Web services. The logic and

algorithms supporting semi-automatic composition belong to this layer. It also includes code for utilizing software in the infrastructure as well as code for serializing the designs and generating design artifacts (BPEL and process WSDL). Since SAWSDL plays a central role in this layer, we begin by presenting it in more detail. We then discuss the components making up this middle layer.

The Semantic Annotations for WSDL and XML Schema (SAWSDL) W3C Recommendation is the first step by W3C to add semantic support to Web service technology by adding semantic annotations to WSDL components. It is now poised to unleash the power, flexibility, and logic of the next generation of Web services. One of the key design principles of SAWSDL is that it enables semantic annotations for Web services using and building on the existing extensibility framework of WSDL. This is the fundamental feature for our framework, since it is incremental (i.e., it will function in a non-semantic world, yet work better as semantics are added).

SAWSDL defines the following three new extensibility attributes for WSDL 2.0 (WSDL 1.1 is also supported) elements to enable semantic annotation of WSDL components:

- An extension attribute, named model Reference, to specify the association between a WSDL component and a concept in some semantic model. This model Reference attribute can be used to annotate XML Schema type definitions, element declarations, and attribute declarations as well as WSDL interfaces, operations, and faults. In our framework, we use model Reference to annotate service operations and XML schema type definitions.
- Two extension attributes, named lifting Schema Mapping and lowering Schema Mapping, that are added to XML Schema element declarations and type definitions for specifying mappings between semantic data and XML. These mappings can be used during service

invocation. (from W3C website). In our framework, we use them to aid data mediation as well.

SAWSDL is easy to learn and use as it extends WSDL. This represents an evolutionary and compatible upgrade of existing Web services standards. Currently, there are already several open-source tools for SAWSDL which are used in our framework, including: SAWSDL4J is an object model for SAWSDL documents. Radiant is a UI for annotating existing WSDL documents based on WSDL-S or SAWSDL via, for example, an OWL Ontology. Lumina is a Semantic Web Service discovery tool that complements Radiant.

There are several components in this Semantic Layer. These components are utilized by both the composition designer (section 6.1) and the discovery tool (section 6.2). A graphical representation is used to save the composition designer's results to disk, which can be done automatically by Eclipse GMF. BPEL serialization uses the NetBeans BPEL API to generate BPEL. The model part of NetBeans SOA project is used as our model implementation. It is an open source project and implements all the features of WS-BPEL.

An important contribution in this project is the design and implementation of the semantic engine. It handles the semantic requests from the UI module, providing information to assist service discovery and data mapping. It connects with the Semantic Web service registry and other components of the Service Infrastructure layer to accomplish its function. Currently, commercial Web service technology lacks full support for service discovery, service invocation/interoperation, service negotiation and service composition. Adding semantics to Web services is the most promising solution for this problem as discussed in section 5. Because of the flexible design of WS-BioZard, it has the potential capability to switch to other semantic

approach. The only modification will be the semantic engine to adopt this change.

Since there are a large number of biological Web services available online, locating the relevant ones manually would be time consuming. Being able to find the most relevant service based on input, output and functionality automatically is very useful. The Lumina Discovery API (see section 6.3) is used to discover Semantic Web Services based upon SAWSDL annotations. The algorithm can find the most relevant Web service through calculating semantic Web service property similarity, based on concepts, semantic relations, and their common and distinguishing features. A Web service's input, output, and functionality are considered during the similarity calculation.

6.3.1. DATA MEDIATION IMPLEMENTATION ISSUES

Once the end user is ready to compose Web services, we have developed an algorithm to find out if two Web services match or not (refer to section 5.2). If the match is close enough, a means for providing missing information can be given manually by the end user at design-time (see the wizard interface in Figure 3.3). The following example shows this process (please refer this example to section 7). Table 3.1 shows the data schema annotated using SAWSDL:

Table 3.1. Data Schema Annotated by SAWSDL

WuBlast Output Schema	<xsd:element name="wuBlastResponse"> <xsd:complexType> <xsd:sequence> <xsd:element name="locations" type="xsd:string" sawsdl:modelReference="http://purl.org/obo/owl/sequence#location_text" /> </xsd:sequence> </xsd:complexType> </xsd:element>

GeneByLocation Input Schema	<xsd:complexType name="inputType"> <xsd:sequence> <xsd:element name="locations" type="xsd:string" sawsdl:modelReference="http://purl.org/obo/owl/sequence#location_text"></xsd:element> <xsd:element name="optionalParameters" type="xsd:string" sawsdl:modelReference="http://purl.org/obo/owl/sequence#OPTIONAL"></xsd:element> <xsd:element name="extension" type="xsd:int" sawsdl:modelReference="http://purl.org/obo/owl/sequence#extent"></xsd:element> </xsd:sequence> </xsd:complexType>

Suppose the user inputs “0” for extension and “Product” for “optionalParameters”, the following <assign> activity will be generated by WS-BioZard:

```
<assign name="invoke102Assign">

 <copy><from><literal>Product</literal></from>
 <to>$invoke103inputVar.parameters/ns102:in/ns102:optionalParameters</to></copy>

 <copy><from><literal>1000</literal></from>
 <to>$invoke103inputVar.parameters/ns102:in/ns102:extension</to></copy>

 <copy><from>$wuBlast102outputVar.parameters/ns101:locations</from>
 <to>$invoke103inputVar.parameters/ns102:in/ns102:locations</to></copy>

</assign>
```

6.3.2. BPEL GENERATION

Considering the complexity of the WS-BPEL specification and the availability of free

open-source BPEL implementations, making our own BPEL model is not a good choice. Currently, there are two free open-source BPEL implementations, NetBeans BPEL and Eclipse BPEL. Both are mature and conform to WS-BPEL 2.0[17]. Although NetBeans BPEL is meant for their own IDE and very hard to separate from their IDE codebase, NetBeans provides a nice editor with automatic execution engine deployment. This feature greatly simplifies BPEL process validation both at design time and run time, which is the main reason why we choose NetBeans BPEL as our BPEL model implementation.

Based on the NetBeans BPEL model, we made a wrapper API corresponding to "BPEL Wrapper API" module, which is a bridge between our BPEL editor and the NetBeans BPEL model. Since we simplify of our BPEL editor (hiding "receive", "reply", "partnerLink", "invoke", "assign" from the end user), we must implement a wrapper/bridge to keep synchronization between the BPEL process and the BPEL model. WS-BioZard also simplifies BPEL deployment via the automatic generation of the process WSDL files as well as NetBeans generated deployment descriptors.

6.4. SERVICE INFRASTRUCTURE LAYER

The semantic layer described in section 6.3 relies on the semantics provided by the bottom layer, the Service Infrastructure layer. There are two main components in this layer: Radiant and Lumina. These tools were initially developed as part of the METEOR-S project [30]. During the standardization process of SAWSDL, they were updated and made generally available on the W3C SAWSDL Website. Radiant is a GUI tool that allows users to rapidly annotate WDSL documents with semantic annotation (model references to ontological concepts) and then publish

17. Although both of them conform to WS-BPEL 2.0, their implementation is slightly different. Hence, not all the BPEL process generated by Eclipse BPEL can be run under NetBeans BPEL.

the annotated services in a semantic registry. Lumina is a tool for discovering semantic Web services and is divided into a GUI part and a discovery API that allows programs to find Web services stored in a semantically enhanced UDDI registry. The discovery API of Lumina is being reused by our discovery tool described in section 6.1.

UDDI discovery is based on syntax and keywords and hence is weak on the use of semantics. Although Lumina can work on regular UDDI registries, improved discovery is provided when semantic annotations are available. Therefore, it is ideal if service providers go to the extra effort to annotate their services following the SAWSDL standard. This procedure provides the key support for service discovery and data mapping. Multiple tools can achieve this task including Radiant and WSMO Studio. Radiant is an open source Eclipse plug-in project developed in the LSDIS lab at the University of Georgia. Since Radiant can not only annotate, but also publish the annotated service to a UDDI registry with the assistance of the discovery API, it is used in our framework.

Lumina's Discovery API [18] provides a semantic layer for Web services on top of a traditional UDDI registry. The API provides semantic equivalents to two of the most important operations in UDDI, publishing and discovering. Publishing is done with Web services that were annotated according to the SAWSDL standard storing all annotations present in the SAWSDL document. The API's discovery mechanism works against an ontology and is able to provide services that are compatible based on reasoning over the ontology.

Ontology plays a fundamental part for semantics by providing the formal definition and relationship between concepts. Because of the complexity of the biological domain, diversity of biological terms and lack of a universal naming convention, ontology is particularly important in the biological domain. For testing our prototype implementation of WS-BioZard, we extended

the well-known Sequence Ontology (SO) (*www.sequenceontology.org/*) as our domain ontology. A joint effort by genome annotation centers, SO includes a set of terms and relationships used to describe the features and attributes of biological sequences. However, our framework can support multiple domain ontologies to provide broader semantics.

6.5. DEPLOYING AND EXECUTING BPEL PROCESSES

Once the completed BPEL specification is created, one needs to deploy it and then execute the BPEL process. Deployment installs the BPEL process on a server hosting a BPEL execution engine. The BPEL engine will treat the BPEL specification like a high-level scripting language, one oriented toward processing XML and invoking services. We have tested our WS-BioZard prototype primarily using Sun's BPEL Server Engine. While many BPEL engines require a deployment descriptor, Sun's BPEL does not, so it simplified the testing of our prototype. It allows automatic deployment to its Glassfish application server.

Once the BPEL specification is deployed, a variety of clients can run the BPEL process. Since the overall BPEL process itself is a regular Web service, it can be invoked by any consumer. Such clients may take the form of a client program or a GUI based Web service invocation tool such as SoapUI (www.soapui.org). To simplify such invocations, we have made a simple Java GUI tool to invoke such Web services.

As a standard Web service, the overall BPEL process must have a WSDL specification to represent it. None of the major, free BPEL editors can automatically generate this WSDL from the BPEL process specification, including the ones we have tried: Active Endpoints' ActiveBPEL Designer, Eclipse BPEL Designer, Sun's NetBeans BPEL Designer, Oracle's JDeveloper BPEL Designer. Considering the complexity of WSDL specifications and their associated XML

schema, manually creating WSDL is challenging for computer scientists, let alone for biologists. However, without this WSDL definition, no BPEL process can be deployed and invoked by any consumer. Since our goal is to substantially reduce the complexity of designing, deploying and executing BPEL processes, WS-BioZard includes the capability to automatically generate WSDL specifications from the BPEL process specification[18]. It parses the BPEL and creates a WSDL operation specification from the information found in the BPEL receive and reply elements as well as variable and namespace specifications.

As an initial step, we annotated multiple bioinformatics Web services using SAWSDL and extended the ontology SO. They are running on our local machine and can be accessed publicly. Two BPEL processes were implemented to verify our prototype implementation. Service providers should annotate their Web services if they want our system to consume their Web services. There are only two things for them to do: First, annotate their Web services' WSDL using SAWSDL. Radiant can help with this task through an intuitive GUI interface. Second, register their Web service to UDDI. Radiant can also help with this task. If the Web services registered in BioMoby are annotated by SAWSDL, we could develop a plugin to communicate with BioMoby's service registration in order to invoke the Web services registered in BioMoby.

7. COMPARISON OF WEB SERVICE COMPOSITION DESIGN TOOLS

In this section, we present a bioinformatics task that requires the composition of two Web services provided by ApiDB: WuBlast and GeneByLocation. The two Web services have been written, semantically annotated and published. The purpose of this example is to compare our system with Taverna, Active-BPEL and NetBeans in facilitating the composition task for the

18. As long as the first BPEL activity is "receive" with "instance=yes" and the last BPEL activity is "reply", WS-BioZard can handle it correctly. This covers most of cases.

user.

The first service "WuBlast" that is located in the LSDIS lab runs the BLAST algorithm developed at Washington University (http://blast.wustl.edu/). BLAST can be used to compare the query sequence to a sequence database and calculate the statistical significance of matches. A BLAST output is a set of matches (or "hits") to sequences contained in a database. These matches are sequences defined by a name and start and stop coordinate values in the genome sequence and are ordered by a confidence score. The matches can help users infer information about the "query sequence" based upon information that is know about the identified hits in the database. Much information is buried in these hits that might be more useful to biologists. For example, the user might want to find genes located either in or near these matching sequences. To accomplish this, we need a second web service "genesByLocation" that tells us, given the hit location (start, stop coordinates), if there are any genes within a user specified distance of the BLAST hit.

Typically a biologist would use multiple web-based tools and a copy/paste approach, as described below:

1. Openning the ApiDB's BLAST web page and the ApiDB's GenesByLocation web page.
2. In the BLAST page, specify the input parameters. Copy one query sequence from the original sequence fasta file and paste it into the "Input Sequence" field. Run BLAST.
3. After getting the BLAST result, the user manually finds out the "best hits" from all the hits, according to some well-defined criteria that measure how identical the matches are, and how well the query sequence length is covered by the matches.
4. The value of "Genomic Sequence Id" should be copied from the previous page and pasted into the corresponding field in the GenesByLocation page. The other two input values,

"start at" and "end location" also come from the BLAST result. The gene sequence might be fully represented within the hit, or it might fall partly within it; the user needs to manually calculate the extension range, a number of nucleotides, which will determine how far beyond the hit start and stop coordinates to search for annotated genes..

5. Copy the gene ids returned and paste/append into a file to be processed further, for example, to obtain the gene protein or nucleotide sequence.

6. Depending on the number of sequences to be located, this process may need to be repeated N times.

From the above procedure, we can see that there are four main problems. Let us suppose that there are N sequences to be processed in the original user supplied fasta sequence file. For each sequence, there are on average M hits and P "best hits". First, there are too many copy/paste user actions. (A) The user needs to copy/paste N*P times from step three to step four. (B) The user needs to copy/save the final results N*P times. Second, too many web page accessions which will cost much time for the end user: N times accessing the web page in the first step. N*P times accessing the web page in the fourth step. Third, the user needs to filter out the "best hits" manually for N*M times. Considering N and M could be very large (hundreds, thousands), one realizes that this is an impossible task for a human being. Fourth, the user needs to manually calculate the "extend" value, the gene range extension beyond the start/end values for the hit, for 2*N*P times.

Compared with the above approach, composing these two services to form a single process can significantly improve efficiency. The biologists no longer will need to save the results of the first step, convert it to the proper format and then carry out the second step. Furthermore, the

process could have been designed with a loop so that many such input sequences could be handled at once. To compose these two services, first we need to filter the BLAST results according to some well-defined criteria. For each hit, the second Web service will identify all the genes within or near it. The user will input a range or extension to determine how far beyond the hit start and stop coordinates to search for annotated genes.

Figure 3.6 shows the BPEL process designed using WS-BioZard. The center part is the design canvas on which users can drag and drop components and visually create the BPEL process. On the top, all the buttons of the toolbar can customize the look of the BPEL process graph such as changing the size, color, font and automatically aligning all the components in the graph. The tools on the top right include zoom, adding notes, and select. Beside these tools is the thumbnail of the BPEL process graph. All the above funcationality is provided by Eclipse GMF. Below the tools section are all the containers which the user can use in BPEL process, such as sequential, if, while and connection. The containers are generated from the EMF model definition. They can be dragged into the canvas to create structures for the BPEL process and then add Web services into these containers. All the available Web services are on the bottom right, so the user can drag and drop the Web service icon to visually compose the BPEL process. Currently, we put the annotated WSDL here. In the future, those Web services would be dynamically retrieved from the UDDI, or dragged from the Service Discovery tool (Section 6.1) When the user clicks any component of the BPEL process on the canvas, the properties of the component will show up in the property section on the bottom right.

To create the BPEL process for the sample workflow we described above, we should drag the “WorkflowBegin” into the canvas. After that, we should add one sequence container and two Web services (wuBlast, geneByLocation) into it. After we add the “WorkflowEnd”, we can use

the “connection” to connect all components together. As soon as we connect the two Web services together, a wizard window will popup (see Figure 3.5) for user to enter values or expressions for the non-matched inputs of the second Web service. After we finish composing the BPEL process, our tool can automatically deploy the BPEL process to Sun’s Glassfish Application Server.

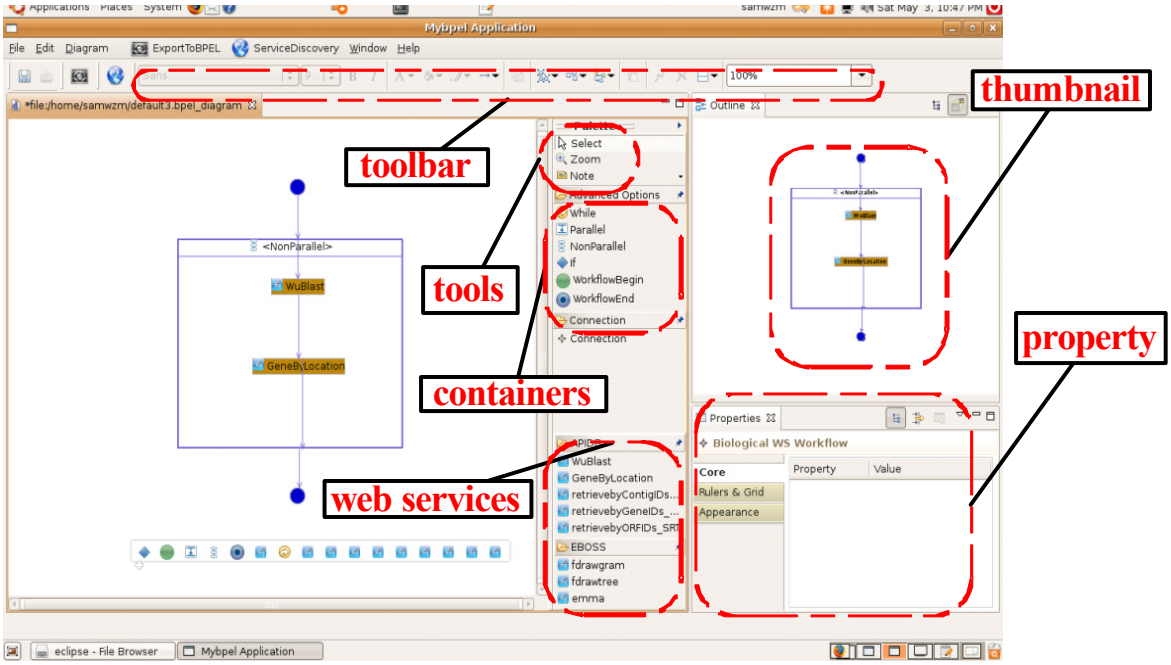

Figure 3.6. Workflow Made by Our BPEL Editor.

Without the help of a service composition tool, the above task would be very time consuming, not only due to the number of service calls required for even a small number of hits, but also considering that one might want to explore different search databases, different filtering criteria and different range values. The following three subsections compare WS-BioZard with three popular alternatives means for composing the same sample workflow: ActiveBPEL Designer (7.1), Sun's NetBeans BPEL Designer (7.2) and Taverna's Designer (7.3). The criteria

for comparison is simply how easy it is for a user to build the workflow and execute it: the fewer steps the better.

7.1. ACTIVEBPEL DESIGNER

The ActiveBPEL Designer, developed by Active Endpoints, Inc, is a visual environment for designing, developing, testing and deploying WS-BPEL 2.0 (BPEL) compliant process definitions. As shown in Figure 3.7, we use the ActiveBPEL Designer 4.1 to compose the sample bioinformatics BPEL process described above. Compared to WS-BioZard, the ActiveBPEL designer provides less automation and is harder for non-computer science users to use.

A BPEL process itself is deployed as a Web service, so a WSDL file has to be created to represent the web service of the BPEL process itself. It is hard for biologists to create such a WSDL file. Our WS-BioZard will automatically create the WSDL file for users, while in the ActiveBPEL Designer users have to manually create this WSDL file. As for data matching/mapping, in our WS-BioZard, this is handled automatically by the data mediation subsystem. While in the ActiveBPEL Designer, users have to do it by themselves, such as define variables, specify the <assign> element to copy matched output of the first Web service to the input of the second Web service. In Figure 3.7, we can see five <assign> elements are used. In Figure 3.6 of WS-BioZard, the user will not be bothered by the <assign> elements; they are generated automatically. Although in the ActiveBPEL Designer, the user can use the wizard to do it, our automatic approach will be easier for biologists. Finially, WS-BioZard can deploy the created BPEL process to server automatically. However, in ActiveBPEL, there are two steps to deploy a BPEL process: First, the user needs to create a Process Deployment Descriptor (.pdd)

file using the wizard. Second, the user needs to export a Business Process Archive (.bpr) file that contains one or more BPEL processes to the execution server.

Comparing Figure 3.7 to Figure 3.6, our BPEL process is much simpler and focuses more on the business flow. Users only have to choose the Web services they want to connect together and WS-BioZard will take care of the details of how to connect them. In Figure 3.7, for the ActiveBPEL Designer users have to understand the details of BPEL specifications such as variables, <assign>/<copy>, XPath and why they need <receive>/<reply> at the ends of the process.

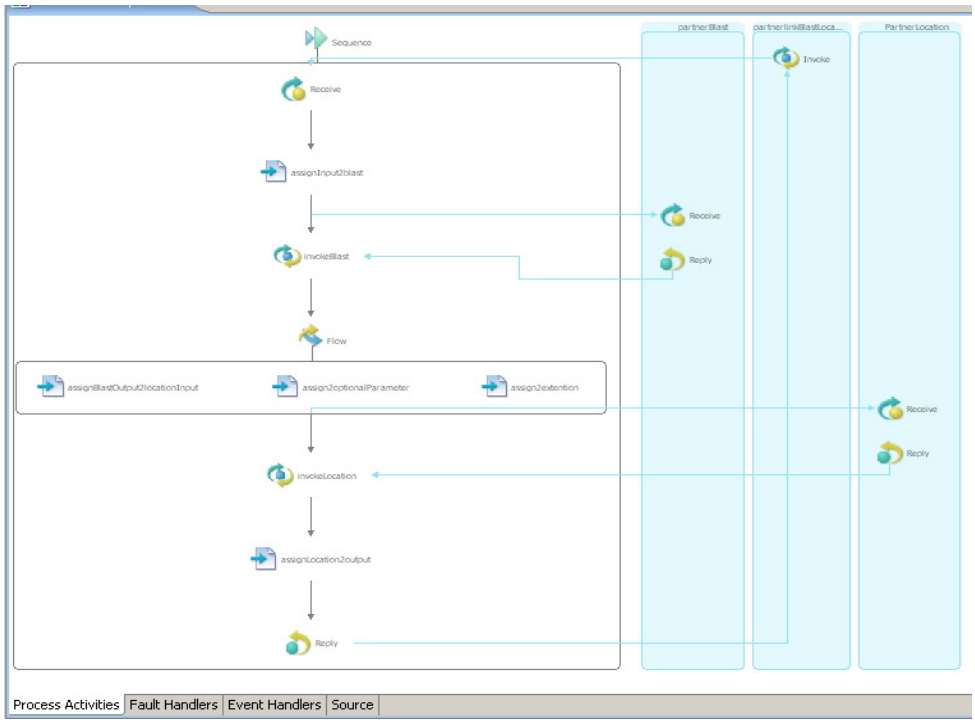

Figure 3.7. Workflow Made by ActiveBPEL.

7.2. SUN'S NETBEANS BPEL DESIGNER

In the NetBeans 6.0.1 release, the BPEL Designer is one component as is shown in Figure 3.8. Similarly to ActiveBPEL, it provides a visual editor to compose and edit BPEL processes.

Our WS-BioZard is more automatic and easier to use compared to NetBeans BPEL Designer, although NetBeans BPEL Designer provides more automation compared to ActiveBPEL.

The NetBeans BPEL Designer cannot automatically generate the WSDL file. Hence, our WS-BioZard, which can automatically create the WSDL file, provides more automation. NetBeans has a GUI tool for users to drag and drop and then generate the XPath used in the <assign> elements. It is helpful for users to manually finish the data mediation when users try to connect two Web services together. However, in WS-BioZard, data mediation is done automatically. As for generate deployment descriptors and deploy BPEL processes, both NetBeans and WS-BioZard can achieve these two tasks automatically and hence provide more automation than ActiveBPEL.

Athough NetBeans has an excellent GUI to help users generate XPath and automatically deploy BPEL processes, users still have to know the details of the BPEL specifications. Thus, NetBeans is similar to ActiveBPEL in this respect and users have to link the input and output data manually, so WS-BioZard which handles data mediation automatically should be easier for biologists to use.

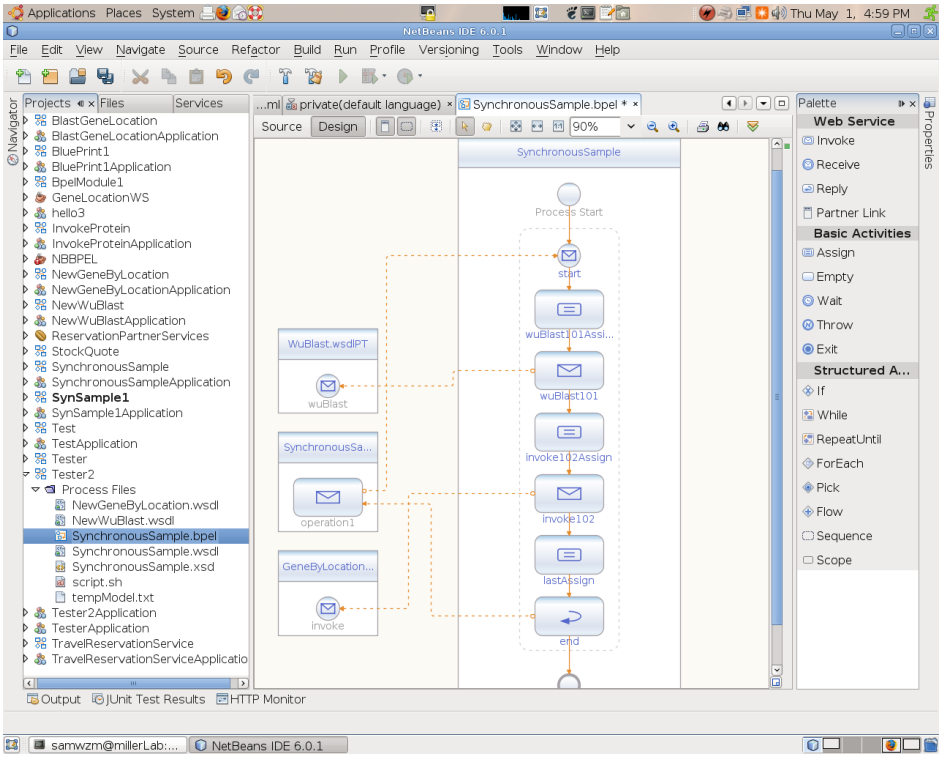

Figure 3.8. Workflow Made by NetBeans.

7.3. TAVERNA'S DESIGNER

The Taverna workbench, created by the myGrid project, is a tool for designing and executing workflows. Taverna allows users to integrate many different software tools, including Web services. The Taverna Workbench provides a desktop authoring environment and execution engine for scientific workflows expressed in XScufl (XML Simple Conceptual Unified Flow language). Therefore, the Taverna Workbench will not create BPEL processes. We compare it here because the Taverna Workbench is a popular tool in the biology and bioinformatics domain.

Since the Taverna Workbench is not a BPEL designer, most of the aspects of automation we compared above are not applicable to Taverna because there are no <assign>, <receive> or <reply> to be generated and no deployment at all. To help handle data mediation, Taverna provides “splitters” to parse the messages exchanged between two Web services. In Figure 3.9,

the two green rectangles are the two Web services that we try to connect together. The purple rectangles are all the splitters. Each splitter only can parse one layer down for one complex type element. That means the number of splitters to use depends on the number of complex type elements in the XML schema for the Web services' messages. Although the splitter is helpful, the biologists still have to have enough XML schema knowledge and manually finish the data mediation task using a scripting language, which is hard for non-programmers. WS-BioZard can handle data mediation automatically without a line of coding, so it has higher automation and is easier to use. As long as the semantic annotation is correct, the WS-BioZard can always automatically generate the correct data mediation.

Since BPEL is the main standard for specifying Web service processes, it is a disadvantage that Taverna does not use it. At some point, they may choose to replace XScufl with BPEL. The two main reasons for not doing this are that, as we have discussed, BPEL is complex and the changeover would require substantial re-coding. Since the workflow made by Taverna workbench is not a Web service, no deployment is required in Taverna, but at the same time it leads to less reusability and accessibility because the result can only be consumed by Taverna. However, WS-BioZard creates the BPEL process which can be deployed as a Web service, which could be invoked/reused by any other Web service or even other BPEL processes.

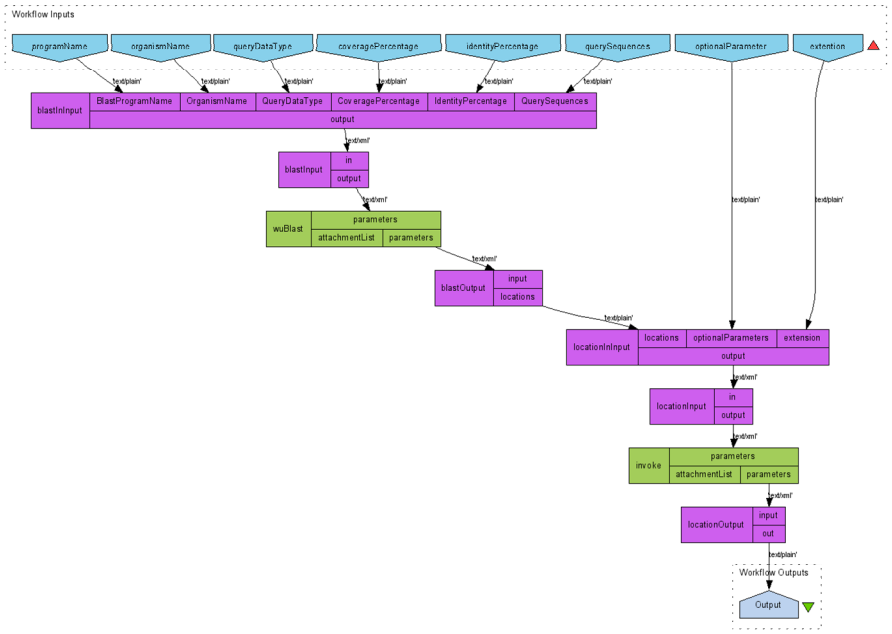

Figure 3.9. Workflow Made by Taverna.

8. CONCLUSIONS AND FUTURE WORK

With more and more bioinformatics Web services available now, the need for composing Web services to achieve complex tasks is increasing. Unfortunately, composing Web services is still too complex to be widely practiced by average biologists. Our framework addresses this problem and uses multiple technologies to lower the complexity of service composition. This framework is predicated on the view that design tools based on the concept of semi-automatic composition can effectively reduce the complexity of building BPEL processes, by utilizing automation where possible and supplementing it with wizards to guide humans in designing the parts that currently do not yield well to automation. The framework is further predicated on the view that providing some level of semantic specification is essential. The new SASWDL standard along with supporting tools such as Radiant and Lumina, provide a low-cost in terms of time and effort, potentially high-gain approach to adding semantics to Web service descriptions,

and thereby facilitating more precise discovery as well as opportunities to automate some aspects of composition.

WS-BioZard is a prototype implementation of this framework, which required the development of new techniques as well as their careful combination with existing Web service technology. An algorithm based on bottom-up semantic annotation of Web services to achieve data mediation automatically was developed. In any composition system, data mediation may be one of the biggest challenges and WS-BioZard provides a practical solution for this challenge. With a growing number of bioinformatics Web services emerging, manually finding and locating a suitable service is not a trivial task. With the aid of the semantic service discovery tool, the end user can query the available Web services published in a semantic UDDI registry through many criteria. WS-BioZard's innovation here was to reuse the discovery API provided with Lumina and develop a new GUI-based front-end that is particularly suitable for biologists, which makes it easy to sift through and visualize how related services can fit together in a composition.

Wizards that guide humans in performing computerized tasks are popular and effective. They are included in WS-BioZard in a parametric way in that their expertise is largely derived from a domain ontology. WS-BioZard includes a multi-faceted wizard serving as a biologist's assistant, helping him/her to design the BPEL process. The use of domain ontology not only provides expertise, but it does so in the language/terminology of the biologist using the tool. Use of the multi-faceted wizard allows a simplified BPEL editor to be presented to the user, and thus lowers the complexity of composing BPEL processes. In this simplified BPEL editor, many BPEL elements such as “variable”, “invoke” ,”assign”, “partnerLink”, “receive” and “reply” are hidden from the users, permitting them to focus more on business logic and not get overwhelmed by low-level details. WS-BioZard can automatically generate XPath expressions to aid in data

mediation and condition expression. XPath is the default language used in BPEL for data access/manipulation and writing XPath manually is both difficult and error-prone.

There are many detailed steps required to make a BPEL process runnable. Every step is required to create the process, so a tool suite for biologists should handle them all. Currently, all of the mainstream BPEL editors use a similar approach—defining the BPEL process WSDL before actually creating the BPEL process or even leaving it up to the designers to create the process WSDL themselves. Since it is not a trivial task for biologists to create a WSDL specification, it is important for WS-BioZard to automatically generate the process WSDL from the BPEL definition. To further simplify BPEL deployment, WS-BioZard can automatically deploy the BPEL process into Sun's Glassfish BPEL execution engine with the help of automatically generated deployment descriptors. Finally, after a BPEL process has been deployed into an execution engine, an intuitive GUI client can be popped-up to invoke the deployed BPEL process as a standard Web service.

With all the advanced features provided by WS-BioZard, such as service discovery, simplified BPEL editor (hides the complexity of the BPEL language from the end user, while keeping most of its functionality), automatic data mediation, a multi-faceted wizard, automatic WSDL generation and deployment, we believe WS-BioZard is easier to use for biologists and could greatly help the complex analysis tasks, especially the tasks which composes multiple steps, need data format convert and have large amount of sequences to process.

In the future, the service discovery tool should be more tightly integrated with the service composition editor. That is, the user should be able to drag-and-drop the service selected in the service discovery tool into the service composition editor. The icons of the discovered services should remain in the composition editor, in a sense, turning the composition problem into a puzzle

solving problem (how to make the color coded services icons fit together). To make WS-BioZard more generally useful, its customization features (particularly for the discovery tool) should be improved and tested on multiple ontologies. Furthermore, a new deployment description could be made to automatically deploy the BPEL process to a central and public accessible Web server. In this way, the end user needs not download and install the web server to further reduce the learning curve for biologists without computer science background. Finally, further wizard capabilities should be added to assist in the creation of more advanced process flow constructs such as automatic loop function as Taverna provides and parallel execution. Once these enhancements have been made, WS-BioZard should be quantitatively evaluated to determine (i) the degree to which it lowers the learning curve for composition, (ii) how much it increases a designer's productivity in creating BPEL processes and (iii) how it helps biologists by saving them time and effort.

ACKNOWLEDGMENTS

Z.W. was supported by NIH R01 AI058515 to J.C.K. We would like to thank all the ApiDB project team members for their help in this study. J.A, J.C.K, D.B and C.A has been funded in whole or in part with Federal funds from the National Institute of Allergy and Infectious Diseases, National Institutes of Health, Department of Health and Human Services, under Contract No. HHSN266200400037C.

REFERENCES

[1] F. Casati, S. Ilnicki, and L. Jin. Adaptive and dynamic service composition in EFlow. In Proceedings of 12th CAiSE, June 2000.

[2] H. Schuster, D. Georgakopoulos, A. Cichocki, and D. Baker. Modeling and composing service-based and reference process-based multi-enterprise processes. In Proceeding of 12th International Conference on Advanced Information Systems Engineering (CAiSE), Stockholm, Sweden, June 2000. Springer Verlag.

[3] F. Casati, M. Sayal, and M.-C. Shan. Developing e-services for composing eservices. In Proceedings of 13th International Conference on Advanced Information Systems Engineering(CAiSE), Interlaken, Switzerland, June 2001. Springer Verlag.

[4] R. Hull, M. Benedikt, V. Christophides, and J. Su. E-service: A look behind the curtain. In Proceeding of the 22th PODS, San Diego, USA, June 2003.

[5] J. Rao and X. Su. "A Survey of Automated Web Service Composition Methods". In Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition, SWSWPC 2004, San Diego, California, USA, July 6th, 2004.

[6] C. Peltz. “Web Services Orchestration and Choreography”, *Web Services Journal*, Volume 03--07, July 2003, pp. 30-35

[7] Ranjit Mulye. “METEOR-S Process Design and Development Tool” Master These

[8] W. M. P. van der Aalst and A. H. M. ter Hofstede. YAWL: yet another workflow language. *Information Systems*. Volume 30, Issue 4, pp. 245-275, 2005.

[9] E. Pignotti, P. Edwards, A. Preece, G. Polhill and N. Gotts. Semantic Workflow Management for E-Social Science. Proceedings of the Third International Conference on eSocial Science, October 2007.

[10] Workflow Patterns Standards Evaluation, www.workflowpatterns.com/evaluations/standard/index.php

[11] C.A. Goble, R. Stevens, G. Ng, S. Bechhofer, N.W. Paton, P.G. Baker, M. Peim, and A. Brass. Transparent Access to Multiple Bioinformatics Information Sources. *IBM Systems Journal*. Special issue on deep computing for the life sciences, 40(2):532 -- 552, 2001.

[12] S. McIlraith, and T. Son. Adapting Golog for Composition of Semantic Web Services. In Proceedings of the 8th International Conference on Principles of Knowledge Representation and Reasoning (KR2002), pp. 482—493, 2002.

[13] Dan Wu, Bijan Parsia , Evren Sirin , James Hendler , and Dana Nau, "Automating DAML-S Web Services Composition Using SHOP2", ISWC 2003, LNCS 2870, pp. 195-210, 2003.

[14] B. Srivastava, J. Koehler, "Web service composition: Current solutions and open problems", ICAPS 2003 Workshop on Planning for Web Services, 28—35, 2003

[15] L. T. Santos, P.A. Roberto, M.A. Gonçalves, H. F. Laender: Design, Implementation, and Evaluation of a Wizard Tool for Setting Up Component-Based Digital Libraries. *Research and Advanced Technology for Digital Libraries*.

[16] C. Petrie. The World Wide Wizard of Open Source Services. 2007, IEEE International Conference on Web Services (ICWS' 2007)

[17] M. Nagarajan, K. Verma, A.P. Sheth and J.A. Miller, "Ontology Driven Data Mediation in Web Services," *International Journal of Web Services Research*, Vol. 4, No. 4. 2007.

[18] K. Verma, K. Sivashanmugam, A.P. Sheth, A. Patil, S. Oundhakar and J.A. Miller, "METEOR-S WSDI: A Scalable P2P Infrastructure of Registries for Semantic Publication and Discovery of Web Services," *Journal of Information Technology and Management*, Special Issue on Universal Global Integration, Vol. 6, No. 1. 2005.

[19] E. Sirin, B. Parsia, and J. Hendler. Filtering and Selecting Semantic Web Services with Interactive Composition Techniques. *IEEE Intelligent Systems*, 19:42–49, 2004.

[20] M. Xu, J. Chen, Y. Peng, X. Mei and Ch. Liu. A Dynamic Semantic Association-Based Web Service Composition Method. Web Intelligence, Issue18-22, 2006.

[21] J. Kim, M. Spraragen, and Y. Gil. An Intelligent Assistant for Interactive Workflow Composition. In IUI ’04: Proceedings of the 9th international conference on Intelligent user interface, pp. 125–131, New York, NY, USA, 2004. ACM Press.

[22] F. Hakimpour, D. Sell, L. Cabral, J. Domingue, and E. Motta. Semantic Web Service Composition in IRS-III: The Structured Approach. In 7th IEEE International Conference on E-Commerce Technology. pp. 484–487, 2005.

[23] R.D. Stevens, A.J. Robinson, and C.A. Goble. my Grid: personalized bioinformatics on the information grid. Bioinformatics, 19(90001):302i–304, 2003.

[24] M.D Wilkinson and M. Links. BioMOBY: an open source biological web services proposal. Brief. Bioinform., 3, 331–341, 2002.

[25] Y.C. Song, E. Kawas, B.M. Good, M.D. Wilkinson and S.J. Tebbutt. DataBiNS: a BioMoby-based data-mining workflow for biological pathways and non-synonymous SNPs. Brief. Bioinform., 23, 780-782, 2007.

[26] Kawas E, et al. BioMoby extensions to the Taverna workflow management and enactment software. *BMC Bioinformatics*, ((2006)) **7**, : 523.

[27] Halevy, A. 2005. Why Your Data Won’t Mix: Semantic Heterogeneity, ACM Queue

[28] Litwin, W. and Abdellatif, A., 1986, Multi-database Interoperability, IEEE Computer, 19(12). 10-18.

[29] Sheth, A., 1998, Changing Focus on Interoperability in Information Systems: From System,

Syntax, Structure to Semantics, Interoperating Geographic Information Systems. 5-30.

[30] Kunal Verma, Kaarthik Sivashanmugam, Amit P. Sheth, Abhijit Patil, Swapna Oundhakar and John A. Miller, "METEOR-S WSDI: A Scalable P2P Infrastructure of Registries for Semantic Publication and Discovery of Web Services," Journal of Information Technology and Management (ITM), Special Issue on Universal Global Integration, Vol. 6, No. 1 (January 2005) pp. 17-39.

[31] Michi Henning, The rise and fall of CORBA, Queue, v.4 n.5, June 2006 [doi>10.1145/1142031.1142044]

[32] R. Chinnici et al., "Web Services Description Language (WSDL) Version 1.2," World Wide Web Consortium, 2002, www.w3.org/TRwsdl12/.

[33] D. Box et al., "Simple Object Access Protocol (SOAP) 1.1," W3C Note 08, World Wide Web Consortium, May 2000, www.w3.org/TRSOAP/.

[34] Muehlen, M., Nickerson, J.V. and Swenson, K.D. (2005), “Developing web services choreography standards – the case of REST vs. SOAP”, Decision Support Systems, Vol. 40, pp. 9-29.

[35] F. Curbera et al., "Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI," IEEE Internet Computing, vol. 6, no. 2, 2002, pp. 86—93.

[36] G. Kandaswamy et al., "Building Web Services for Scientific Applications," IBM J. Research and Development, vol. 50, no. 2/3, 2006, pp. 249–260.

[37] Ping Jiang , Quentin Mair , Julian Newman, Using UML to Design Distributed Collaborative Workflows: from UML to XPDL, Proceedings of the Twelfth International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, p.71, June 09-11, 2003

[38] F. Leymann, Web Services Flow Language (WSFL) 1.0. IBM Software Group, May 2001.

[39] S. Thatte XLANG: Web Services For Business Process Design, Microsoft Corporation (2001)

[40] S. Narayanan, S.A. McIlraith, Simulation, verification and automated composition of web services, in: Proceedings of the Eleventh International World Wide Web Conference (WWW-11), Honolulu, Hawaii, USA, May 2002.

[41] Aurrecoechea, C., Heiges, M., Wang, H., Wang, Z., Fischer, S., Rhodes, P., et al. (2007) ApiDB: integrated resources for the apicomplexan bioinformatics resource center. Nucleic Acids Res 35: D427–D430.

[42] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, D.F. Ferguson (Editors). Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More. Prentice Hall, 2005.

[43] A. Barros, M. Dumas, and P. Oaks. A critical overview of the web services choreography description language(ws-cdl). Business Process Trends, 2005.

[44] A. Arkin. Business Process Modeling Language (BPML), Version 1.0. BPML.org, 2002.

[45] OMG UML 2.0 Specifications. http://www.omg.org/uml/

[46] E. Newcomer and G. Lomow. Understanding SOA with Web Services. International Thomson Computer Press, 2004.

[47] James Clark and Steve DeRose (eds.). XML path language (XPath) version 1.0. W3C Recommendation.

[48] Zixin Wu, Karthik Gomadam, Ajith Ranabahu, Amit P. Sheth and John A. Miller. Automatic Composition of Semantic Web Services using Process and Data Mediation. Proceedings of the 9-th (ICEIS'07), pp. 453-461.

[49] Robert D. Stevens, Alan J. Robinson and Carole A. Goble**.** myGrid: personalised bioinformatics on the information grid. Bioinformatics Vol. 19 Suppl. 1 2003.

APPENDIX: MODERN GUI DEVELOPMENT USING THE ECLIPSE PLATFORM

Creating a modern Graphical User Interface (GUI) with a full complement of features can be very time consuming and often requires upwards of 10,000 lines of code. Fortunately, development frameworks have matured enough to permit the development of advanced GUI's much more rapidly. Several toolkits and frameworks from the Eclipse platform were utilized in the development of WS-BioZard. These are grouped into two levels: The low-level user interface technologies used were the Eclipse's Standard Widget Toolkit (SWT), JFace and Graphical Editing Framework (GEF). The high level technologies used were the Eclipse Modeling Framework (EMF) and Eclipse Graphical Modeling Framework (GMF).

The Eclipse EMF project is a modeling framework that provides a code generation facility. The model can be defined by XML, UML, Java classes or Ecore, providing a Java code generation function. It can be used in many domains including BPEL (e.g., the Eclipse BPEL project uses Eclipse EMF to create their model). Figure 3.10 shows the classes used by our BPEL editor generated by Eclipse EMF.

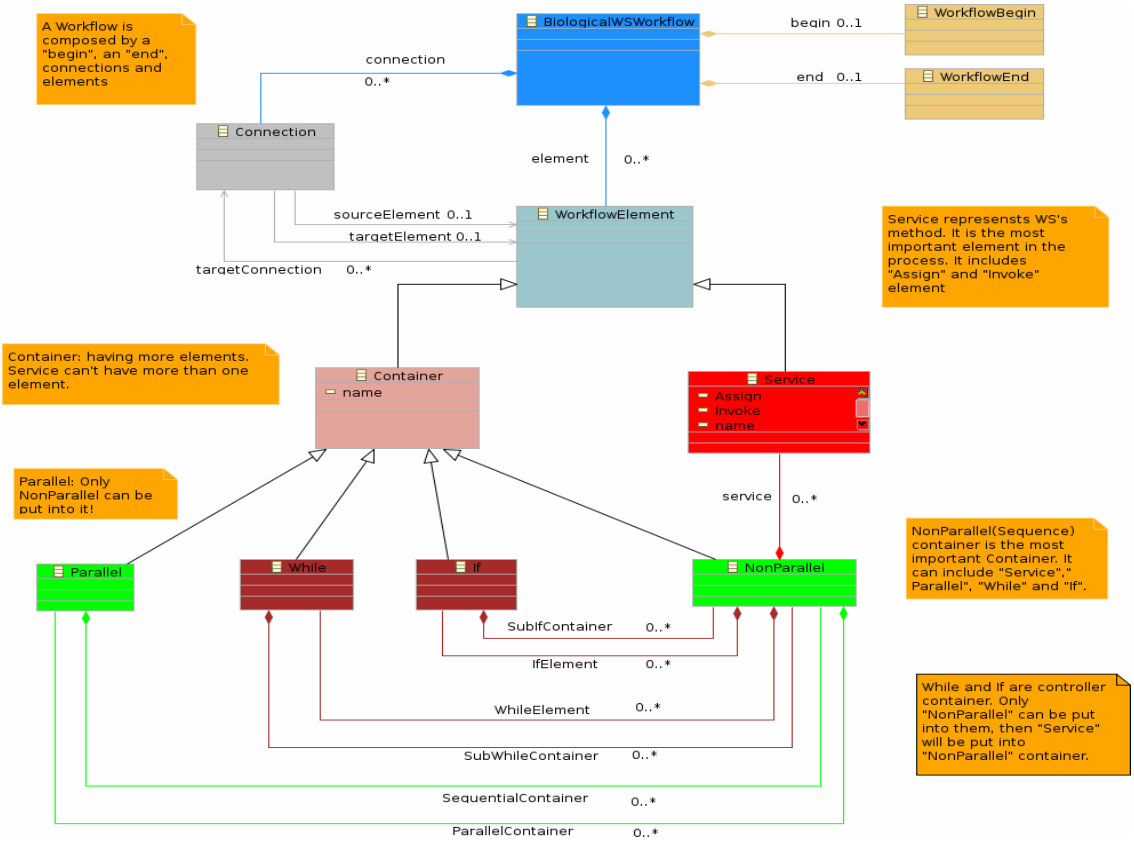

Figure 3.10. BPEL Editor Classes Generated by Eclipse EMF.

Eclipse GEF uses many design patterns, such as Model-View-Controller (MVC), Command, and Chain of Responsibility, to provide quite powerful and flexible functionality. The Eclipse GMF project aims to improve the usability of GUI applications through combining Eclipse EMF with GEF. In this visual development approach, much of the source code will be automatically generated based on the model and can also be customized by the developer.

Using these new high-level Eclipse frameworks, a developer can create a sophisticated GUI with a fraction of the effort that would have been required in the past. This also allows the developer to focus on the application logic and on making software easy to maintain and extend.

This infrastructure can provide the following benefits:

- Focus more on high level concerns—since low level graphical part can be automatically generated by GMF.
- Faster development—utilizing the visual design environment and source code generation.
- Fewer bugs—by reducing manual coding compared with pure Eclipse GEF development.
- Improved maintenance—facilitated by well organized package structure.
- Easy extension both on function part and behavior part—for function part, only thing to do is modify the GMF model definition and regenerate source code; for behavior part, it is also easy to achieve thanks to the design patterns used in EMF.

CHAPTER 4

CONCLUSIONS

Biologists blessed with a huge amount of available bioinformatics data are often confronted with the problems of having to manually identify proper databases and integrate query results when their tasks involve multiple sources. To provide better query facilities and expedite the research process, database integration is an essential technology and has been one of the most important bioinformatics research areas. For the ApiDB project, several approaches for integrating multiple existing bioinformatics databases are investigated, including federated databases, data warehousing, JDBC, database links, Web services and Java RMI. Using JDBC for database federation is a desirable solution, especially when maintaining local autonomy is an important factor. However, the database federation approaches are not easy to extend compared with the Web services approach. Because of platform independence, language independence and decoupling of service from the client, the Web services approach provides the most flexibility, particularly when the system needs to be extended to incorporate more resources. In the past, the performance of Web service technology had been a concern for many developers. Based on our performance evaluation of response time for the eight different implementations we tested, the latest generation of Web service technology (e.g., Axis2) performs very competitively. Considering the flexibility and competitive performance of the Web service approach, it has the potential to provide an excellent solution for biological database integration problems. Furthermore, based on experience from the ApiDB project, it can meet the requirements for

practical industrial strength database integration.

With more and more bioinformatics Web services now available, the need for composing Web services to achieve complex tasks is increasing. Unfortunately, for biologists the learning curve to compose Web services is still too high to be widely practiced. To lower the complexity of composing Web services, semi-automatic composition techniques can be applied to effectively reduce the difficulty of building BPEL processes compared with the typical manual approach. Our approach utilizes automation where possible and supplements it with wizards to guide humans in designing the parts that currently do not yield well to automation. We developed the WS-BioZard prototype to provide a simplified way for biologists or bioinformaticians to compose Web services.

In any composition system, data mediation may be one of the biggest challenges and WS-BioZard provides a practical solution for this. An algorithm based on bottom-up semantic annotation of Web services to achieve data mediation automatically, was developed. WS-BioZard can automatically generate XPath expressions that go into the BPEL process specification. XPath is the default language used in BPEL for data access/manipulation and writing XPath manually is both difficult and error-prone. WS-BioZard also includes a multi-faceted wizard serving as a biologist's assistant, facilitating the end user to design the BPEL process. With the help of automatic data mediation and the wizard, the simplified BPEL designer provided by WS-BioZard can greatly lower the complexity of composing bioinformatics Web services. To assist with composition, WS-BioZard includes a semantic Web service discovery tool, enabling users to query the available Web services published in a semantic UDDI registry. Through reusing the discovery API provided with Lumina (part of the METEOR-S project), we developed a new GUI-based front-end that is particularly suitable for

biologists. The service discovery tool makes it easy to sift through and visualize how related services can fit together in a composition.

In order to invoke a BPEL process as a standard Web service, a WSDL file must first be generated from the BPEL process specification, and then the BPEL process must be deployed into a BPEL execution engine. Both steps can be challenging, especially for an initiated biologists. Currently, all of the mainstream BPEL editors use a similar approach—defining the BPEL process WSDL before actually creating the BPEL process or even leaving it up to the designers to create the process WSDL themselves. WS-BioZard can automatically handle both steps which greatly simplifies the BPEL deployment process. To further simplify BPEL invocation, after a BPEL process has been deployed into an execution engine, an intuitive GUI client can be popped-up to invoke the deployed BPEL process as a standard Web service.

REFERENCES

[1] How Federated Databases Benefit Bioinformatics Research (http://www.b-eye-network.com/view/2164)

[2] Stein, L. (2003). Integrating biological databases. Nat. Rev. Gen. 4, 337-345.

[3] Stevens, R., Goble, C.A., Baker, P. & Brass, A. (2001). A classification of tasks in bioinformatics. Bioinformatics 17, 180–188.

[4] Sheth, A. & Larson, J.A. (1990). Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. ACM Computing Surveys 22 (3), 183-236.

[5] Karp, P. (1995). A strategy for database interoperation. J. Comput.Biol., 2, 573–586.

APPENDICES

A. INSTALLATION GUIDE

WS-BioZard is released as a stand alone application which can run under Linux, Unix and Macintosh operationing systems. WS-BioZard is released under the GNU GPL v3 open source license. The application may be downloaded based on your operating system at the following URLs (Goto http://lsdis.cs.uga.edu/projects/meteor-s/downloads/ first):

Linux with GTK:

X86: linux.gtk.x86.zip

PPC: linux.gtk.ppc.zip

Mac OSX:

Carbon x86: macosx.carbon.x86.zip

Carbon PPC: macosx.carbon.ppc.zip

Unix (Solaris): solaris.gtk.sparc.zip

WS-BioZard has the following dependencies: Java 1.6 or greater (http://java.sun.com/javase/downloads/index.jsp) and the Sun Glassfish Application Server for running BPEL Processes (https://glassfish.dev.java.net/). Detailed installation instructions for each Java 1.6 and Glassfish are provided when you download each package. Simple installation instructions are provided here:

Java 1.6 for Linux Users:

Download the JRE 1.6 or JDK 1.6 for Linux (the 32 bit VM will run on even on

64bit Linux installs if you are unsure if you have a 64bit OS.)

Navigate to the location where you saved the “bin” file

Make sure the file is executable “chmod 0755 <bin file name>”

Execute the bin file (e.g. “./jdk-1.6_u10.bin”)

If you downloaded the RPM version, install the RPM with your package manager

If you downloaded the regular bin file, add the <extracted java dir>/bin to your path

Glassfish:

http://docs.sun.com/app/docs/prod/iss.trad#hic

WS-BioZard:

Download the Application

Unzip the Application to a directory of your choice

B. USERS GUIDE

After unzipping the file if you did not change the directory name, you will find the executables for each version in the following directories:

Linux version: there is a runnable file “biozard” under the directory of “linux.gtk.x86/eclipse” or “linux.gtk.ppc/eclipse”.

Macintosh version: there is a runnable file “biozard.app” under the directory of “macosx.carbon.x86/eclipse” or “macosx.carbon.ppc/eclipse”.

Unix version: there is a runnable file “biozard” under the directory of “solaris.gtk.sparc/eclipse/”.

First, the splash screen will pop-up with a status bar to indicate the start progress of WS-BioZard. If this is the first time running WS-BioZard, a window with the error message “Cannot open input element” will pop-up as shown in Figure B-1. The error is caused by the inability to find the files “default.bpel_diagram” and “default.bpel”. Those two files are used by Eclipse EMF to store model information. “default.bpel_diagram” is used to store the information about the picture on the canvas as shown in Table B-1. The “default.bpel” file is used to store information about the model as shown in Table B-2.

Table B-1. Content of “default.bpel_diagram”

```
<?xml version="1.0" encoding="UTF-8"?>

<notation:Diagram xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:mybpel="lime.ctegd.uga.edu" xmlns:notation="http://www.eclipse.org/gmf/runtime/1.0.1/notation"
xmi:id="_YRyfsCgwEd2Hoq4ONr2N8g" type="Mybpel" name="default2.bpel_diagram" measurementUnit="Pixel">

  <children xmi:type="notation:Node" xmi:id="_4CxlECgwEd2Hoq4ONr2N8g" type="2002">

    <styles xmi:type="notation:ShapeStyle" xmi:id="_4CxlESgwEd2Hoq4ONr2N8g" fontName="Sans"/>

    <element xmi:type="mybpel:WorkflowBegin" href="default.bpel#//@workflowElements.0"/>

    <layoutConstraint xmi:type="notation:Bounds" xmi:id="_4CxlEigwEd2Hoq4ONr2N8g" x="440" y="55"/>

  </children>
```

```
  <children xmi:type="notation:Node" xmi:id="_5Dmv8CgwEd2Hoq4ONr2N8g" type="2004">
    <children xmi:type="notation:Node" xmi:id="_5DxvECgwEd2Hoq4ONr2N8g" type="5003"/>
    <children xmi:type="notation:Node" xmi:id="_5D0yYCgwEd2Hoq4ONr2N8g" type="7009">
      <children xmi:type="notation:Node" xmi:id="_Cyt5kCgxEd2Hoq4ONr2N8g" type="3009">
        <children xmi:type="notation:Node" xmi:id="_Cy5fwCgxEd2Hoq4ONr2N8g" type="5015"/>
        <styles xmi:type="notation:ShapeStyle" xmi:id="_Cyt5kSgxEd2Hoq4ONr2N8g" fontName="Sans"/>
        <element xmi:type="mybpel:Service" href="default.bpel#//@workflowElements.1/@services.0"/>
        <layoutConstraint xmi:type="notation:Bounds" xmi:id="_Cyt5kigxEd2Hoq4ONr2N8g" x="138" y="61"/>
      </children>
      <children xmi:type="notation:Node" xmi:id="_JkDDcCgxEd2Hoq4ONr2N8g" type="3010">
        <children xmi:type="notation:Node" xmi:id="_JkRs8CgxEd2Hoq4ONr2N8g" type="5016"/>
        <styles xmi:type="notation:ShapeStyle" xmi:id="_JkDDcSgxEd2Hoq4ONr2N8g" fontName="Sans"/>
        <element xmi:type="mybpel:Service" href="default.bpel#//@workflowElements.1/@services.1"/>
        <layoutConstraint xmi:type="notation:Bounds" xmi:id="_JkDDcigxEd2Hoq4ONr2N8g" x="127" y="237"/>
      </children>
      <styles xmi:type="notation:SortingStyle" xmi:id="_5D0yYSgwEd2Hoq4ONr2N8g"/>
      <styles xmi:type="notation:FilteringStyle" xmi:id="_5D0yYigwEd2Hoq4ONr2N8g"/>
    </children>
    <styles xmi:type="notation:ShapeStyle" xmi:id="_5DnXACgwEd2Hoq4ONr2N8g" fontName="Sans"/>
    <element xmi:type="mybpel:NonParallel" href="default.bpel#//@workflowElements.1"/>
    <layoutConstraint xmi:type="notation:Bounds" xmi:id="_5DnXASgwEd2Hoq4ONr2N8g" x="231" y="143" width="452" height="408"/>
  </children>
  <children xmi:type="notation:Node" xmi:id="_5oT_gCgwEd2Hoq4ONr2N8g" type="2005">
    <styles xmi:type="notation:ShapeStyle" xmi:id="_5oT_gSgwEd2Hoq4ONr2N8g" fontName="Sans"/>
    <element xmi:type="mybpel:WorkflowEnd" href="default.bpel#//@workflowElements.2"/>
    <layoutConstraint xmi:type="notation:Bounds" xmi:id="_5oT_gigwEd2Hoq4ONr2N8g" x="455" y="655"/>
  </children>
  <styles xmi:type="notation:DiagramStyle" xmi:id="_YRyfsSgwEd2Hoq4ONr2N8g"/>
  <element xmi:type="mybpel:BiologicalWSWorkflow" href="default.bpel#/"/>
  <edges xmi:type="notation:Edge" xmi:id="_RcZfwCgxEd2Hoq4ONr2N8g" type="4001" source="_4CxlECgwEd2Hoq4ONr2N8g"
target="_5Dmv8CgwEd2Hoq4ONr2N8g">
    <styles xmi:type="notation:RoutingStyle" xmi:id="_RcZfwSgxEd2Hoq4ONr2N8g"/>
    <styles xmi:type="notation:FontStyle" xmi:id="_RcZfwigxEd2Hoq4ONr2N8g" fontName="Sans"/>
    <element xsi:nil="true"/>
    <bendpoints xmi:type="notation:RelativeBendpoints" xmi:id="_RcZfwygxEd2Hoq4ONr2N8g" points="[5, 10, 0, -70]$[5, 78, 0, -2]"/>
```

```
    <targetAnchor xmi:type="notation:IdentityAnchor" xmi:id="_Rdg6ECgxEd2Hoq4ONr2N8g" id="(0.49557522,0.004901961)"/>

  </edges>

  <edges xmi:type="notation:Edge" xmi:id="_XOQwsCgxEd2Hoq4ONr2N8g" type="4001" source="_5Dmv8CgwEd2Hoq4ONr2N8g" target="_Cyt5kCgxEd2Hoq4ONr2N8g">

    <styles xmi:type="notation:RoutingStyle" xmi:id="_XOQwsSgxEd2Hoq4ONr2N8g"/>

    <styles xmi:type="notation:FontStyle" xmi:id="_XOQwsigxEd2Hoq4ONr2N8g" fontName="Sans"/>

    <element xsi:nil="true"/>

    <bendpoints xmi:type="notation:RelativeBendpoints" xmi:id="_XOQwsygxEd2Hoq4ONr2N8g" points="[-7, -5, -5, -98]$[-84, 90, -82, -3]"/>

    <sourceAnchor xmi:type="notation:IdentityAnchor" xmi:id="_XOUbECgxEd2Hoq4ONr2N8g" id="(0.5022124,0.012254902)"/>

  </edges>

  <edges xmi:type="notation:Edge" xmi:id="_lOS88CgxEd2Hoq4ONr2N8g" type="4001" source="_Cyt5kCgxEd2Hoq4ONr2N8g" target="_JkDDcCgxEd2Hoq4ONr2N8g">

    <styles xmi:type="notation:RoutingStyle" xmi:id="_lOS88SgxEd2Hoq4ONr2N8g"/>

    <styles xmi:type="notation:FontStyle" xmi:id="_lOS88igxEd2Hoq4ONr2N8g" fontName="Sans"/>

    <element xsi:nil="true"/>

    <bendpoints xmi:type="notation:RelativeBendpoints" xmi:id="_lOS88ygxEd2Hoq4ONr2N8g" points="[5, 10, 1, -166]$[5, 166, 1, -10]"/>

  </edges>

  <edges xmi:type="notation:Edge" xmi:id="_pvcz8CgxEd2Hoq4ONr2N8g" type="4001" source="_JkDDcCgxEd2Hoq4ONr2N8g" target="_5Dmv8CgwEd2Hoq4ONr2N8g">

    <styles xmi:type="notation:RoutingStyle" xmi:id="_pvcz8SgxEd2Hoq4ONr2N8g"/>

    <styles xmi:type="notation:FontStyle" xmi:id="_pvcz8igxEd2Hoq4ONr2N8g" fontName="Sans"/>

    <element xsi:nil="true"/>

    <bendpoints xmi:type="notation:RelativeBendpoints" xmi:id="_pvcz8ygxEd2Hoq4ONr2N8g" points="[0, -10, 13, 258]$[-13, -274, 0, -6]"/>

    <targetAnchor xmi:type="notation:IdentityAnchor" xmi:id="_pvhscCgxEd2Hoq4ONr2N8g" id="(0.4778761,0.014705882)"/>

  </edges>

  <edges xmi:type="notation:Edge" xmi:id="_uJZPsCgxEd2Hoq4ONr2N8g" type="4001" source="_5Dmv8CgwEd2Hoq4ONr2N8g" target="_5oT_gCgwEd2Hoq4ONr2N8g">

    <styles xmi:type="notation:RoutingStyle" xmi:id="_uJZPsSgxEd2Hoq4ONr2N8g"/>

    <styles xmi:type="notation:FontStyle" xmi:id="_uJZPsigxEd2Hoq4ONr2N8g" fontName="Sans"/>

    <element xsi:nil="true"/>

    <bendpoints xmi:type="notation:RelativeBendpoints" xmi:id="_uJZPsygxEd2Hoq4ONr2N8g" points="[2, 402, 3, -114]$[9, 516, 10, 0]"/>

    <sourceAnchor xmi:type="notation:IdentityAnchor" xmi:id="_uJevQCgxEd2Hoq4ONr2N8g" id="(0.5199115,0.014705882)"/>

  </edges>

</notation:Diagram>
```

Table B-2. Content of “default.bpel”

```
<?xml version="1.0" encoding="UTF-8"?>

<mybpel:BiologicalWSWorkflow xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:mybpel="lime.ctegd.uga.edu">

  <workflowElements xsi:type="mybpel:WorkflowBegin" childNodes="//@workflowElements.1"/>

  <workflowElements xsi:type="mybpel:NonParallel" childNodes="//@workflowElements.1/@services.0 //@workflowElements.2">

    <services childNodes="//@workflowElements.1/@services.1" name="wuBlast_WuBlast_APIDB" type="wuBlast_WuBlast_APIDB"/>

    <services childNodes="//@workflowElements.1" name="wuBlastFilter_WuBlast_APIDB" type="wuBlastFilter_WuBlast_APIDB"/>

  </workflowElements>

  <workflowElements xsi:type="mybpel:WorkflowEnd"/>

</mybpel:BiologicalWSWorkflow>
```

After clicking “OK” to dismiss the error message two files “default.bpel_diagram” and “default_bpel” are created. The Web service editor of WS-BioZard will now pop-up, Figure B-2. The error seen when the Web service editor opens is the UI reporting the same error seen in Figure B-1 again. After clicking “OK” and creating new files of “default.bpel_diagram” and “default.bpel” through clicking “File”-->”New”, the blank Web service editor of WS-BioZard will be shown as Figure B-3. If WS-BioZard was run before, there should be no error messages as in Figure B-1 and B-2. Instead, the application will go to Figure B-3 directly. In the future, this problem could be solved by switching the default GEF default file to the empty template file.

As shown in Figure B-6, WS-BioZard is composed of a menu&tool bar (on the top), a canvas used to show/edit the model (in the middle), a palette, a thumbnail, and a property viewer.

- Menu&tool bar: The service discovery tool can be invoked by clicking “ServiceDiscovery” icon. Tools used to aid BPEL process editing are also in tool bar, such as font color, font size, font style, elements auto-alignment.
- Palette: there are four groups: the top one is system tools, such as “Select” used to select

particular elements, “Zoom” used to zoom in/out the BPEL process. The next group is BPEL elements including process begin/end, If, While, Parallel, NonParallel (The reason why we use NonParallel instead of Sequence is because sequence in biology domain usually means DNA/RNA or protein sequence, which has totally different meaning with the usage in BPEL process. To avoid confusion, we chose the term “NonParallel” to represent a sequential activities in BPEL process). The third group is Connect, which is used to connect elements in the BPEL process. The bottom group is the Bio-Informatics Web services categorized by service provider. Each one represents a single method/operation instead of a Web service. (One Web service could contain multiple methods/operations.)

- Thumbnail: If the BPEL process is very big and complex, the end user can browse the different parts of the BPEL process conveniently through this function.

In order to automatically generate a WSDL file from a BPEL process, we use the creation of a connection as the trigger. Whenever there a connection is created, a corresponding event will be triggered and executed. For example, if a connection from process beginning to the main sequential block is created, a directory storing the BPEL process file and the WSDL files will be created. A window also pops-up to tell the end user where the files are stored. Whenever a connection between the main sequential block with process end is created, four things will happen in order:

First, the BPEL process file will automatically generate.

Second, a WSDL file representing this BPEL process will automatically generate.

Third, this BPEL process will automatically deploy to Sun Glassfish BPEL execution engine.

Fourth, a client GUI will automatically popup to aid the end user to invoke this newly deployed BPEL process as a standard Web service.

From above, we can see the connection order is very important. The connection between process beginning with the main sequential block should be always the first connection to create. The connection between the main sequential block with the process ending should be always the last connection to create.

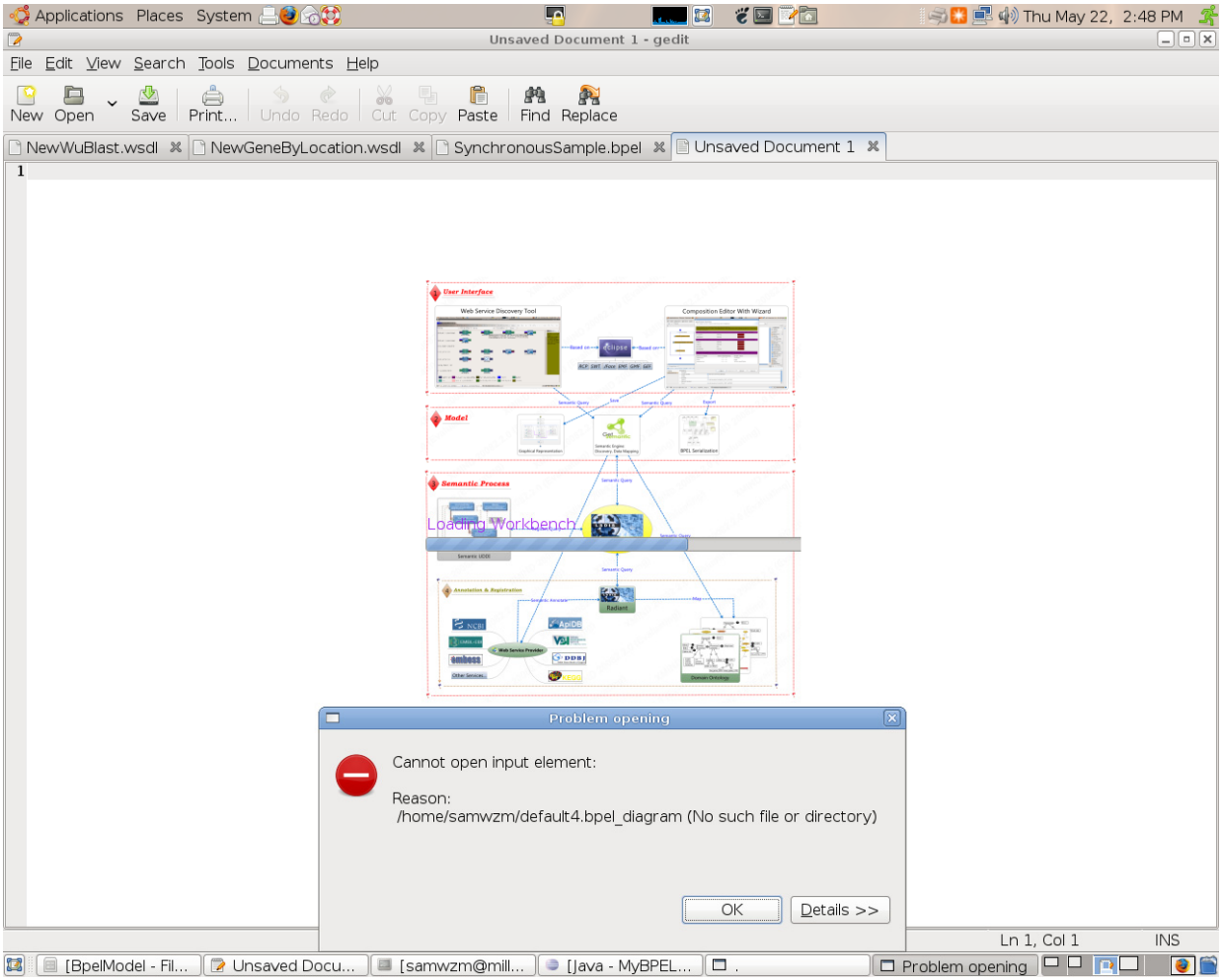

Figure B-1. First Running Error Message 1.

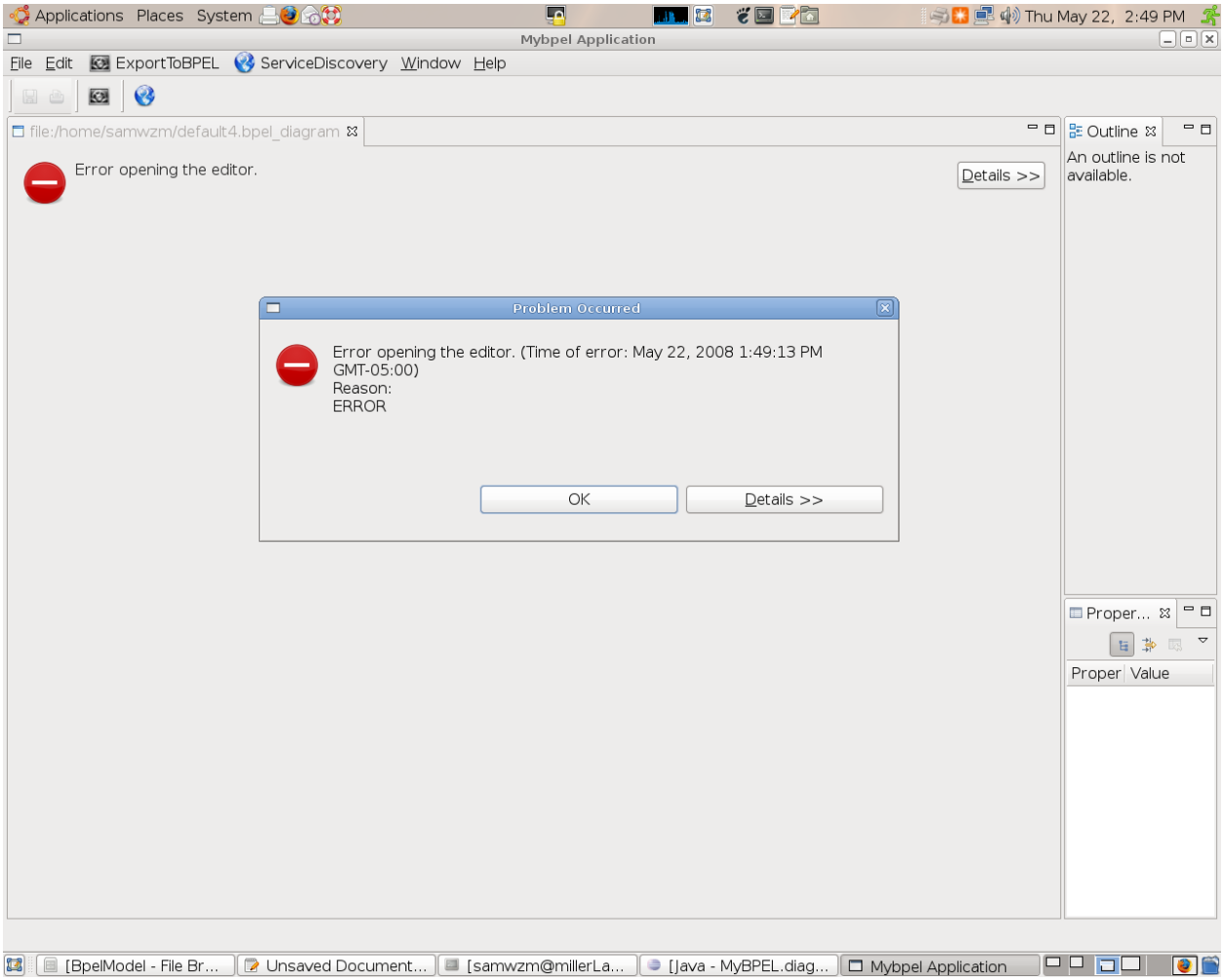

Figure B-2. First Running Error Message 2.

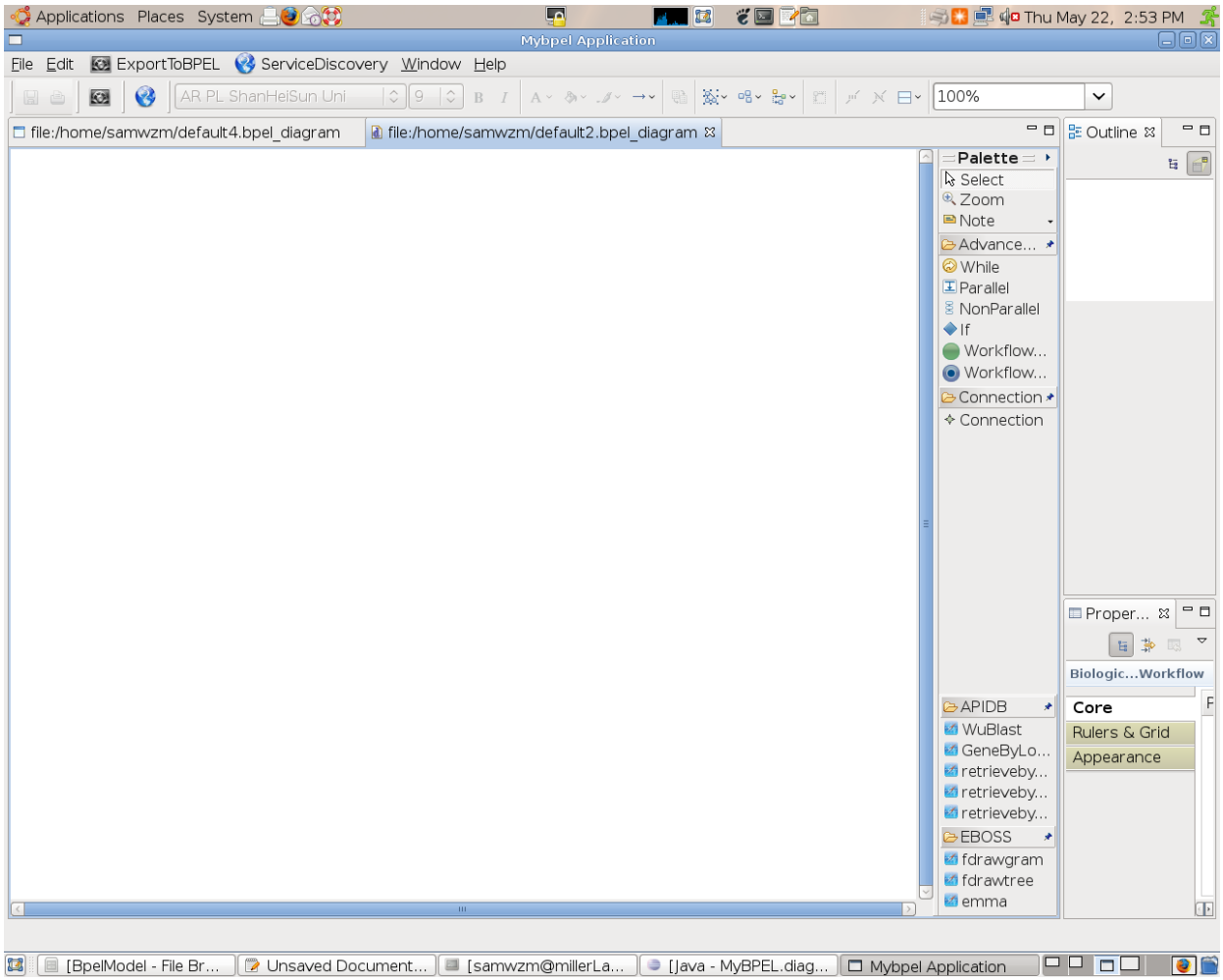

Figure B-3. Blank Web Service Composition Editor.

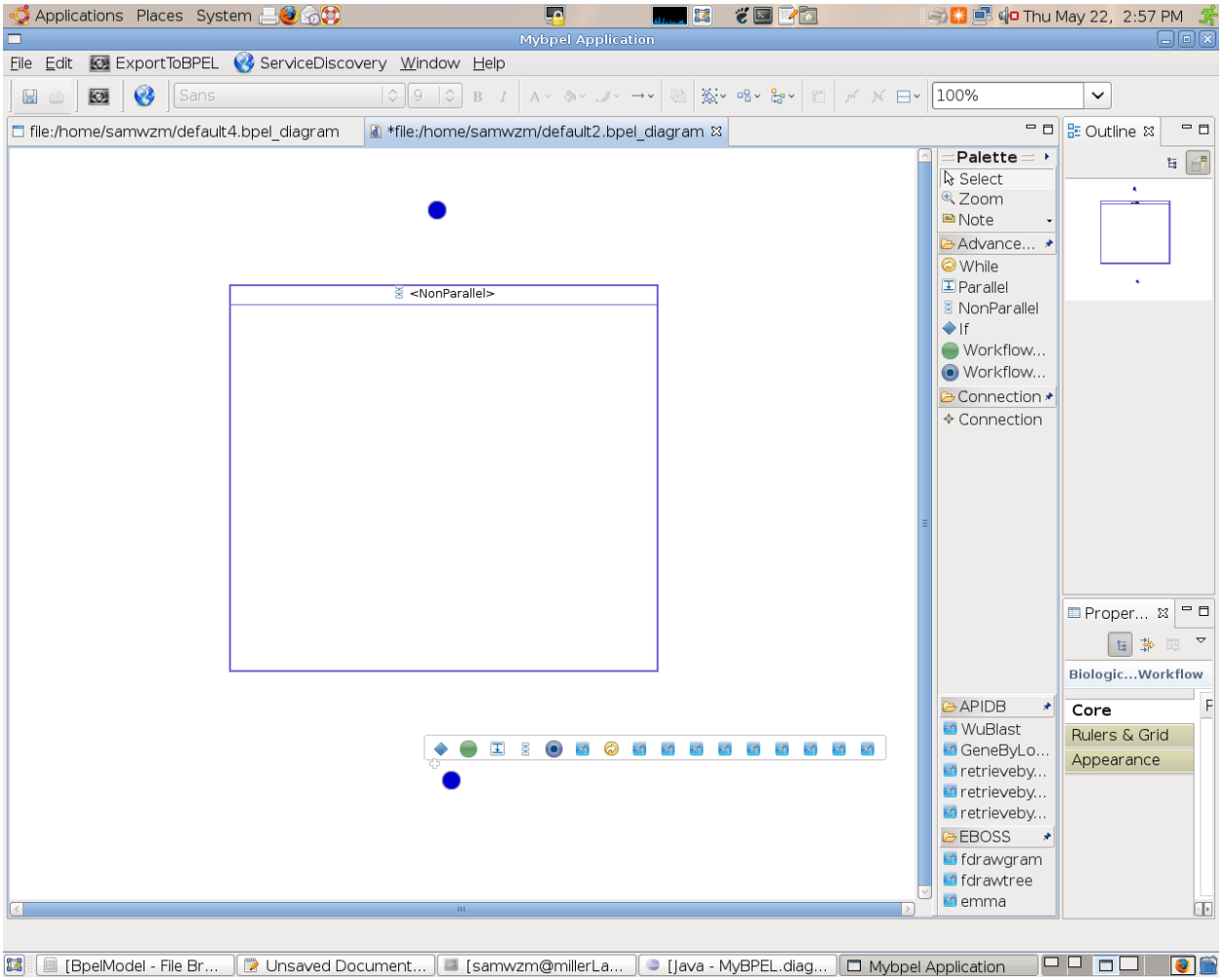

Figure B-4. BPEL Process Skeleton.

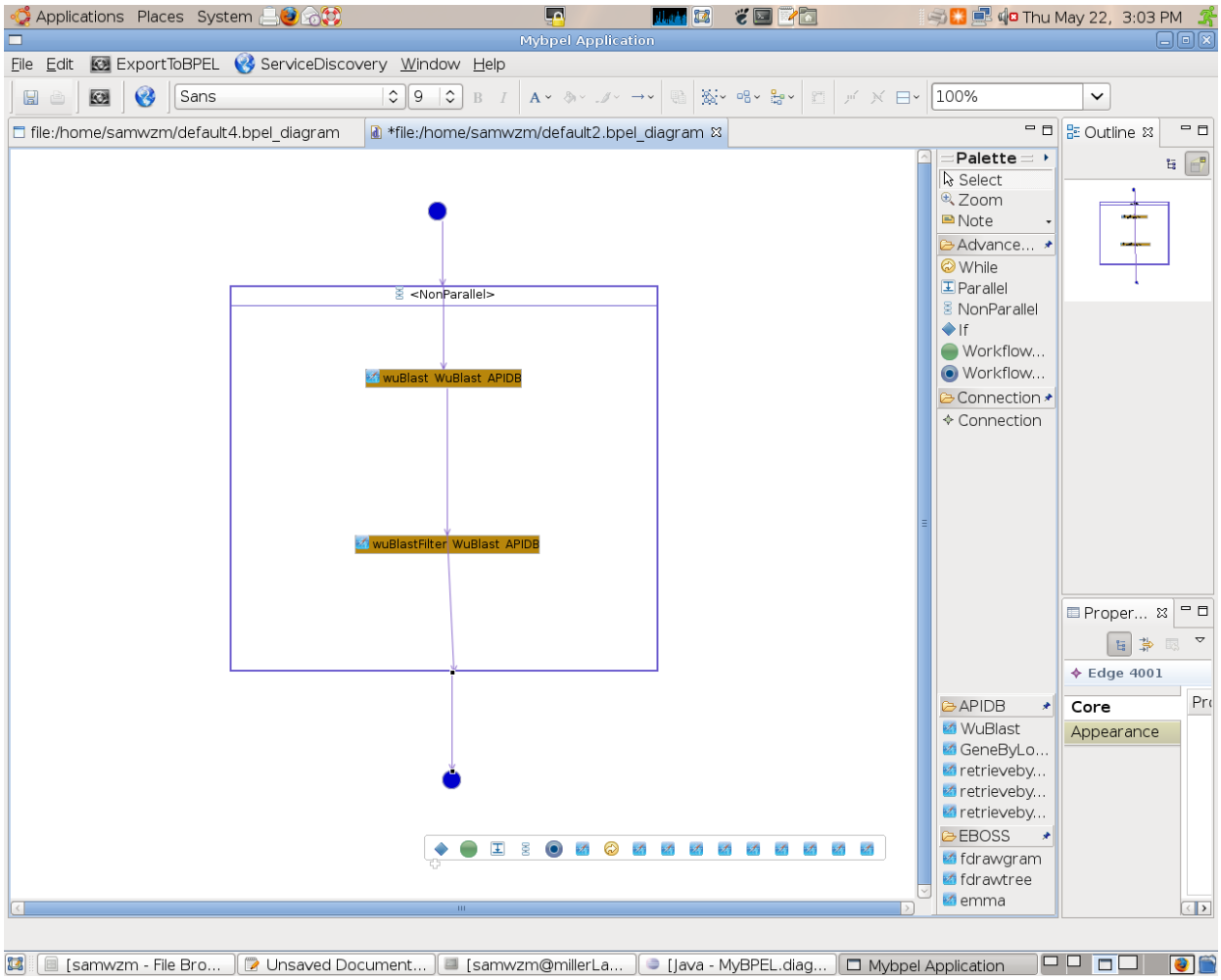

Figure B-5. Final BPEL Process.

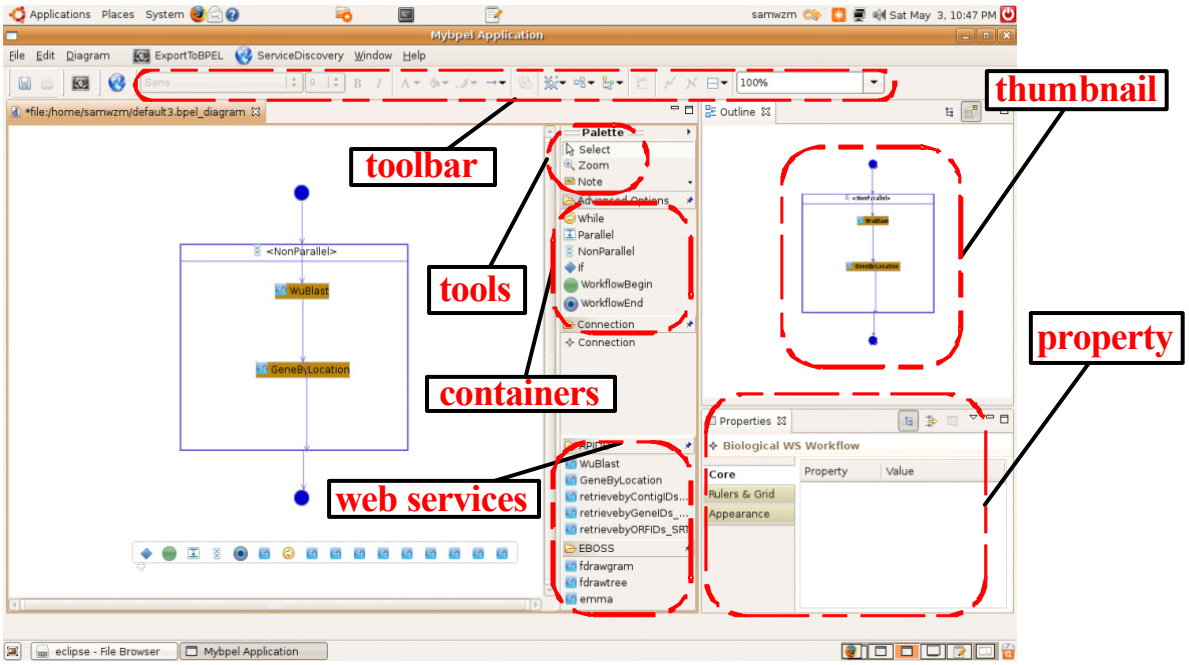

Figure B-6. BPEL Editor Interface.

C. DEVELOPERS GUIDE -- APIS

There are three modules in WS-BioZard, service discovery module, composition editor module and BPEL model module. The Java Docs of the three modules are too big to be put here. They can be downloaded from the following URL as the reference for further development:

http://lsdis.cs.uga.edu/projects/metor-s/downloads/

service discovery module: Service-discovery-doc.zip

composition editor module: Editor-doc.zip

BPEL model module: Model-doc.zip

D. SCENARIO IN DETAIL INCLUDING BPEL, SAWSDL AND WSDL FILES

WuBlast WSDL with SAWSDL Annotation:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
     xmlns:sawsdl="http://www.w3.org/ns/sawsdl"
     xmlns:tns="http://lime.ctegd.uga.edu/NewWuBlast/"
     xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="NewWuBlast"
     targetNamespace="http://lime.ctegd.uga.edu/NewWuBlast/"
     xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype">
     <wsdl:types>
          <xsd:schema elementFormDefault="qualified"
               targetNamespace="http://lime.ctegd.uga.edu/NewWuBlast/">
               <xsd:element name="wuBlast">
                    <xsd:complexType>
                         <xsd:sequence>
                              <xsd:element name="in"
                                   type="tns:inputComplexType" />
                         </xsd:sequence>
                    </xsd:complexType>
               </xsd:element>
               <xsd:element name="wuBlastResponse">
                    <xsd:complexType>
                         <xsd:sequence>
                              <xsd:element name="locations" type="xsd:string"
                                   sawsdl:modelReference="http://purl.org/obo/owl/sequence#location_text" />
                         </xsd:sequence>
                    </xsd:complexType>
               </xsd:element>

               <xsd:complexType name="inputComplexType">
                    <xsd:sequence>
                         <xsd:element name="BlastProgramName"
                              type="xsd:string">
                         </xsd:element>
                         <xsd:element name="OrganismName"
                              type="xsd:string">
                         </xsd:element>
                         <xsd:element name="QueryDataType"
                              type="xsd:string">
                         </xsd:element>
                         <xsd:element name="CoveragePercentage"
                              type="xsd:float">
                         </xsd:element>
                         <xsd:element name="IdentityPercentage"
                              type="xsd:float">
                         </xsd:element>
                         <xsd:element name="QuerySequences"
                              type="xsd:string">
                         </xsd:element>
                    </xsd:sequence>
               </xsd:complexType>
          </xsd:schema>
     </wsdl:types>
     <wsdl:message name="wuBlastRequest">
          <wsdl:part element="tns:wuBlast" name="parameters" />
     </wsdl:message>
```

```
    <wsdl:message name="wuBlastResponse">
        <wsdl:part element="tns:wuBlastResponse" name="parameters" />
    </wsdl:message>
    <wsdl:portType name="NewWuBlast">
        <wsdl:operation name="wuBlast">
            <wsdl:input message="tns:wuBlastRequest" />
            <wsdl:output message="tns:wuBlastResponse" />
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="NewWuBlastSOAP" type="tns:NewWuBlast">
        <soap:binding style="document"
            transport="http://schemas.xmlsoap.org/soap/http" />
        <wsdl:operation name="wuBlast">
            <soap:operation
                soapAction="http://lime.ctegd.uga.edu/NewWuBlast/wuBlast" />
            <wsdl:input>
                <soap:body use="literal" />
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal" />
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="NewWuBlast">
        <wsdl:port binding="tns:NewWuBlastSOAP" name="NewWuBlastSOAP">
            <soap:address
                location="http://128.192.251.200:8080/axis2/services/NewWuBlast" />
        </wsdl:port>
    </wsdl:service>

    <plnk:partnerLinkType name="WuBlastLinkType">
        <plnk:role name="WuBlastRole" portType="tns:NewWuBlast" />
    </plnk:partnerLinkType>

</wsdl:definitions>
```

GeneByLocation WSDL with SAWSDL Annotation:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:sawsdl="http://www.w3.org/ns/sawsdl"
    xmlns:tns="http://www.example.org/NewGeneByLocation/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="NewGeneByLocation"
    targetNamespace="http://www.example.org/NewGeneByLocation/"
    xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype">
    <wsdl:types>
        <xsd:schema elementFormDefault="qualified"
            targetNamespace="http://www.example.org/NewGeneByLocation/">
            <xsd:element name="invoke">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="in" type="tns:inputType">
                        </xsd:element>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="invokeResponse">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="out" type="xsd:string" />
```

```
                    </xsd:sequence>
               </xsd:complexType>
          </xsd:element>

          <xsd:complexType name="inputType">
               <xsd:sequence>
                    <xsd:element name="locations" type="xsd:string"
                         sawsdl:modelReference="http://purl.org/obo/owl/sequence#location_text">
                    </xsd:element>
                    <xsd:element name="optionalParameters"
                         type="xsd:string"
                         sawsdl:modelReference="http://purl.org/obo/owl/sequence#OPTIONAL">
                    </xsd:element>
                    <xsd:element name="extension" type="xsd:int"
                         sawsdl:modelReference="http://purl.org/obo/owl/sequence#extent">
                    </xsd:element>
               </xsd:sequence>
          </xsd:complexType>
     </xsd:schema>
</wsdl:types>
<wsdl:message name="invokeRequest">
     <wsdl:part element="tns:invoke" name="parameters" />
</wsdl:message>
<wsdl:message name="invokeResponse">
     <wsdl:part element="tns:invokeResponse" name="parameters" />
</wsdl:message>
<wsdl:portType name="NewGeneByLocation">
     <wsdl:operation name="invoke">
          <wsdl:input message="tns:invokeRequest" />
          <wsdl:output message="tns:invokeResponse" />
     </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="NewGeneByLocationSOAP"
     type="tns:NewGeneByLocation">
     <soap:binding style="document"
          transport="http://schemas.xmlsoap.org/soap/http" />
     <wsdl:operation name="invoke">
          <soap:operation
               soapAction="http://www.example.org/NewGeneByLocation/invoke" />
          <wsdl:input>
               <soap:body use="literal" />
          </wsdl:input>
          <wsdl:output>
               <soap:body use="literal" />
          </wsdl:output>
     </wsdl:operation>
</wsdl:binding>
<wsdl:service name="NewGeneByLocation">
     <wsdl:port binding="tns:NewGeneByLocationSOAP"
          name="NewGeneByLocationSOAP">
          <soap:address
               location="http://128.192.251.200:8080/axis/services/NewGeneByLocation" />
     </wsdl:port>
</wsdl:service>
<plnk:partnerLinkType name="NewGeneByLocationLinkType">
     <plnk:role name="NewGeneByLocationRole"
          portType="tns:NewGeneByLocation" />
</plnk:partnerLinkType>
</wsdl:definitions>
```

BPEL Process Generated by WS-BioZard:

```
<?xml version="1.0" encoding="UTF-8"?>
<process
     xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
     xmlns:bpws="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
     xmlns:ns1="http://localhost/SynchronousSample/SynchronousSample"
     xmlns:ns101="http://lime.ctegd.uga.edu/NewWuBlast/"
     xmlns:ns102="http://www.example.org/NewGeneByLocation/"
     xmlns:ns2="http://xml.netbeans.org/schema/SynchronousSample"
     xmlns:wsdlNS="http://enterprise.netbeans.org/bpel/SynchronousSample/SynchronousSample_1"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="SynchronousSample"
     targetNamespace="http://enterprise.netbeans.org/bpel/SynchronousSample/SynchronousSample_1">

     <import importType="http://schemas.xmlsoap.org/wsdl/"
          location="NewGeneByLocation.wsdl"
          namespace="http://www.example.org/NewGeneByLocation/" />
     <import importType="http://schemas.xmlsoap.org/wsdl/"
          location="NewWuBlast.wsdl"
          namespace="http://lime.ctegd.uga.edu/NewWuBlast/" />
     <import importType="http://www.w3.org/2001/XMLSchema"
          location="SynchronousSample.xsd"
          namespace="http://xml.netbeans.org/schema/SynchronousSample" />
     <import importType="http://schemas.xmlsoap.org/wsdl/"
          location="SynchronousSample.wsdl"
          namespace="http://localhost/SynchronousSample/SynchronousSample" />

     <partnerLinks>
          <partnerLink myRole="partnerlinktyperole1"
               name="SynchronousSample" partnerLinkType="ns1:partnerlinktype1" />
          <partnerLink xmlns:ns0="http://lime.ctegd.uga.edu/NewWuBlast/"
               name="NewWuBlast.wsdlPT" partnerLinkType="ns0:WuBlastLinkType"
               partnerRole="WuBlastRole" />
          <partnerLink
               xmlns:ns0="http://www.example.org/NewGeneByLocation/"
               name="NewGeneByLocation.wsdlPT"
               partnerLinkType="ns0:NewGeneByLocationLinkType"
               partnerRole="NewGeneByLocationRole" />
     </partnerLinks>

     <variables>
          <variable messageType="ns1:responseMessage" name="outputVar" />
          <variable messageType="ns1:requestMessage" name="inputVar" />
          <variable xmlns:ns0="http://lime.ctegd.uga.edu/NewWuBlast/"
               messageType="ns0:wuBlastRequest" name="wuBlast101inputVar" />
          <variable xmlns:ns0="http://lime.ctegd.uga.edu/NewWuBlast/"
               messageType="ns0:wuBlastResponse" name="wuBlast102outputVar" />
          <variable xmlns:ns0="http://www.example.org/NewGeneByLocation/"
               messageType="ns0:invokeRequest" name="invoke103inputVar" />
          <variable xmlns:ns0="http://www.example.org/NewGeneByLocation/"
               messageType="ns0:invokeResponse" name="invoke104outputVar" />
     </variables>

     <sequence>
          <receive createInstance="yes" name="start"
               operation="operation1" partnerLink="SynchronousSample"
               portType="ns1:portType1" variable="inputVar">
          </receive>
          <assign name="wuBlast101Assign">
               <copy>
                    <from>
                         $inputVar.inputType/ns1:in/ns1:BlastProgramName
                    </from>
```

```
                <to>
                    $wuBlast101inputVar.parameters/ns101:in/ns101:BlastProgramName
                </to>
            </copy>
            <copy>
                <from>
                    $inputVar.inputType/ns1:in/ns1:IdentityPercentage
                </from>
                <to>
                    $wuBlast101inputVar.parameters/ns101:in/ns101:IdentityPercentage
                </to>
            </copy>
            <copy>
                <from>$inputVar.inputType/ns1:in/ns1:OrganismName</from>
                <to>
                    $wuBlast101inputVar.parameters/ns101:in/ns101:OrganismName
                </to>
            </copy>
            <copy>
                <from>
                    $inputVar.inputType/ns1:in/ns1:QueryDataType
                </from>
                <to>
                    $wuBlast101inputVar.parameters/ns101:in/ns101:QueryDataType
                </to>
            </copy>
            <copy>
                <from>
                    $inputVar.inputType/ns1:in/ns1:QuerySequences
                </from>
                <to>
                    $wuBlast101inputVar.parameters/ns101:in/ns101:QuerySequences
                </to>
            </copy>
            <copy>
                <from>
                    $inputVar.inputType/ns1:in/ns1:CoveragePercentage
                </from>
                <to>
                    $wuBlast101inputVar.parameters/ns101:in/ns101:CoveragePercentage
                </to>
            </copy>
        </assign>
        <invoke xmlns:ns0="http://lime.ctegd.uga.edu/NewWuBlast/"
            inputVariable="wuBlast101inputVar" name="wuBlast101"
            operation="wuBlast" outputVariable="wuBlast102outputVar"
            partnerLink="NewWuBlast.wsdlPT" portType="ns0:NewWuBlast" />
        <assign name="invoke102Assign">
            <copy>
                <from>
                    <literal>Product</literal>
                </from>
                <to>
                    $invoke103inputVar.parameters/ns102:in/ns102:optionalParameters
                </to>
            </copy>
            <copy>
                <from>
                    <literal>0</literal>
                </from>
                <to>
```

```
                    $invoke103inputVar.parameters/ns102:in/ns102:extension
               </to>
          </copy>
          <copy>
               <from>
                    $wuBlast102outputVar.parameters/ns101:locations
               </from>
               <to>
                    $invoke103inputVar.parameters/ns102:in/ns102:locations
               </to>
          </copy>
     </assign>
     <invoke xmlns:ns0="http://www.example.org/NewGeneByLocation/"
          inputVariable="invoke103inputVar" name="invoke102" operation="invoke"
          outputVariable="invoke104outputVar"
          partnerLink="NewGeneByLocation.wsdlPT"
          portType="ns0:NewGeneByLocation" />
     <assign name="lastAssign">
          <copy>
               <from>$invoke104outputVar.parameters/ns102:out</from>
               <to>$outputVar.resultType/ns1:out</to>
          </copy>
     </assign>
     <reply name="end" operation="operation1"
          partnerLink="SynchronousSample" portType="ns1:portType1"
          variable="outputVar" />
</sequence>
</process>
```

WSDL File Automatically Generated from the Above BPEL Process:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="SynchronousSample"
     targetNamespace="http://localhost/SynchronousSample/SynchronousSample"
     xmlns:tns="http://localhost/SynchronousSample/SynchronousSample"
     xmlns:bpws="http://docs.oasis-open.org/wsbpel/2.0/varprop"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:ns="http://localhost/SynchronousSample/SynchronousSample"
     xmlns:plink="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
     xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
     xmlns="http://schemas.xmlsoap.org/wsdl/">
     <wsdl:types>
          <xsd:schema elementFormDefault="qualified"
               targetNamespace="http://localhost/SynchronousSample/SynchronousSample"
               xmlns:tns50="http://localhost/SynchronousSample/SynchronousSample">
               <xsd:element name="wuBlast">
                    <xsd:complexType>
                         <xsd:sequence>
                              <xsd:element name="in"
                                   type="tns50:inputComplexType" />
                         </xsd:sequence>
                    </xsd:complexType>
               </xsd:element>
               <xsd:complexType name="inputComplexType">
                    <xsd:sequence>
                         <xsd:element name="BlastProgramName"
                              type="xsd:string">
                         </xsd:element>
                         <xsd:element name="OrganismName"
```

```
                    type="xsd:string">
                </xsd:element>
                <xsd:element name="QueryDataType"
                    type="xsd:string">
                </xsd:element>
                <xsd:element name="CoveragePercentage"
                    type="xsd:float">
                </xsd:element>
                <xsd:element name="IdentityPercentage"
                    type="xsd:float">
                </xsd:element>
                <xsd:element name="QuerySequences"
                    type="xsd:string">
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
        <xsd:element name="invokeResponse">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element name="out" type="xsd:string" />
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
        <xsd:complexType name="inputType">
            <xsd:sequence>
                <xsd:element name="locations"
                    sawsdl:modelReference="http://purl.org/obo/owl/sequence#location_text"
                    type="xsd:string" xmlns:sawsdl="http://www.w3.org/ns/sawsdl" />
                <xsd:element name="optionalParameters"
                    sawsdl:modelReference="http://purl.org/obo/owl/sequence#OPTIONAL"
                    type="xsd:string" xmlns:sawsdl="http://www.w3.org/ns/sawsdl">
                </xsd:element>
                <xsd:element name="extension"
                    sawsdl:modelReference="http://purl.org/obo/owl/sequence#extent"
                    type="xsd:int" xmlns:sawsdl="http://www.w3.org/ns/sawsdl" />
            </xsd:sequence>
        </xsd:complexType>
    </xsd:schema>
</wsdl:types>
<wsdl:message name="responseMessage">
    <wsdl:part name="resultType" element="tns:invokeResponse">
    </wsdl:part>
</wsdl:message>
<wsdl:message name="requestMessage">
    <wsdl:part name="inputType" element="tns:wuBlast"></wsdl:part>
</wsdl:message>
<wsdl:portType name="portType1">
    <wsdl:operation name="operation1">
        <wsdl:input name="input1" message="tns:requestMessage">
        </wsdl:input>
        <wsdl:output name="output1" message="tns:responseMessage">
        </wsdl:output>
    </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="binding1" type="tns:portType1">
    <soap:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="operation1">
        <wsdl:input name="input1">
            <soap:body use="literal" />
        </wsdl:input>
```

```
            <wsdl:output name="output1">
                <soap:body use="literal" />
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="service1">
        <wsdl:port name="port1" binding="tns:binding1">
            <documentation />
            <soap:address
                location="http://localhost:18181/SynchronousSample" />
        </wsdl:port>
    </wsdl:service>
    <plink:partnerLinkType name="partnerlinktype1">
        <plink:role name="partnerlinktyperole1"
            portType="tns:portType1" />
    </plink:partnerLinkType>
</wsdl:definitions>
```